

x/Open[®] Portability Guide

*XSI Commands
and Utilities*



x/Open[®]

x/Open Portability Guide

X/Open represents a major breakthrough in the world of Open Systems. A large number of the world's major information system suppliers, supported by representatives of the user, system integrator, and software development communities, have come together to make true Open Systems a practical reality.

This is achieved by the establishment of a comprehensive integrated **Common Applications Environment**, which ensures portability and connectivity of applications and allows users to move between systems without retraining.

X/Open's objectives are:

- Portability of applications at the source code level so that application porting is a mechanical recompilation process.
- Connectivity of applications via portable networking services that are independent of underlying protocols, plus support for common protocol stacks to ensure that X/Open machines may be interconnected.
- A consistent approach to the user interface with the system.

X/Open is not a standards-setting body. It is a joint initiative by members of the business community to adopt and adapt existing standards into a consistent environment. Where there is an agreed official standard, X/Open adopts it; where there is no agreed official standard, X/Open adopts de facto standards where it is necessary to provide the comprehensive environment demanded by users and applications developers.

The X/Open Portability Guide contains an evolving portfolio of practical integrated standards for application portability. All X/Open members guarantee to support the defined interfaces.

PRENTICE HALL, Englewood Cliffs, N.J. 07632

ISBN 0-13-685635-X

X/Open Portability Guide

XSI Commands and Utilities

X/Open Company, Ltd.

DISCARD



PRENTICE HALL, Englewood Cliffs, New Jersey 07632



© 1989, X/Open Company Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

X/Open Portability Guide

Set of 7 Volumes

ISBN: 0-13-685819-8

Volume 1	XSI Commands and Utilities	ISBN: 0-13-685835-X
Volume 2	XSI System Interfaces and Headers	ISBN: 0-13-685843-0
Volume 3	XSI Supplementary Definitions	ISBN: 0-13-685850-3
Volume 4	Programming Languages	ISBN: 0-13-685868-6
Volume 5	Data Management	ISBN: 0-13-685876-7
Volume 6	Window Management	ISBN: 0-13-685884-8
Volume 7	Networking Services	ISBN: 0-13-685892-9

Printed and Published by Prentice-Hall, Inc., U.S.A.

Set in Palatino, Helvetica and courier by The Instruction Set Ltd., U.K.

Any comments relating to the material contained in the X/Open Portability Guide Issue 3 may be submitted to the X/Open Group at:

X/Open Company Limited
Abbots House
Abbey Street
Reading
Berkshire, RG1 3BD
United Kingdom

or by Electronic Mail to:

xpg3@xopen.co.uk

Contents

XSI COMMANDS AND UTILITIES

Chapter	1	INTRODUCTION
	1.1	OVERVIEW
	1.1.1	Rationale
	1.1.2	Important Note
	1.2	STATUS OF INTERFACES
	1.2.1	Mandatory
	1.2.2	Optional
	1.2.3	Development
	1.2.4	New Interfaces
	1.2.5	Withdrawn
	1.2.6	Portability
	1.3	RELATIONSHIP TO OTHER STANDARDS
	1.4	FORMAT OF ENTRIES
	1.4.1	Typographical Conventions
Chapter	2	DEFINITIONS
	2.1	GLOSSARY
	2.1.1	Absolute Pathname
	2.1.2	Access Mode
	2.1.3	Address Space
	2.1.4	Appropriate Privileges
	2.1.5	Background Process Group
	2.1.6	Block Special File
	2.1.7	C Standard
	2.1.8	Character
	2.1.9	Character Sets
	2.1.10	Character Special File
	2.1.11	Child Process
	2.1.12	Clock Tick
	2.1.13	Command Interpreter
	2.1.14	Controlling Process
	2.1.15	Controlling Terminal
	2.1.16	Current Working Directory
	2.1.17	Device
	2.1.18	Device ID
	2.1.19	Directory

2.1.20	Directory Entry
2.1.21	Directory Stream
2.1.22	Dot
2.1.23	Dot-dot
2.1.24	Effective Group ID
2.1.25	Effective User ID
2.1.26	Empty Directory
2.1.27	Empty String
2.1.28	Epoch
2.1.29	Extended Security Controls
2.1.30	Feature Test Macro
2.1.31	FIFO Special Files (or FIFO)
2.1.32	File
2.1.33	File Access Permissions
2.1.34	File Description
2.1.35	File Descriptor
2.1.36	File Group Class
2.1.37	File Hierarchy
2.1.38	File Mode
2.1.39	Filename
2.1.40	File Offset
2.1.41	File Other Class
2.1.42	File Owner Class
2.1.43	File Permission Bits
2.1.44	File Serial Number
2.1.45	File System
2.1.46	File Times Update
2.1.47	Foreground Process Group
2.1.48	Foreground Process Group ID
2.1.49	Group ID
2.1.50	Job Control
2.1.51	Link
2.1.52	Link Count
2.1.53	Message Catalogue
2.1.54	Message Catalogue Descriptor
2.1.55	Mode
2.1.56	NaN (Not a Number)
2.1.57	Null Character
2.1.58	Null Pointer
2.1.59	Null String
2.1.60	Open File
2.1.61	Open File Description
2.1.62	Orphaned Process Group
2.1.63	Parent Directory
2.1.64	Parent Process
2.1.65	Parent Process ID
2.1.66	Path Prefix

Contents

2.1.67	Pathname
2.1.68	Pathname Component
2.1.69	Pathname Resolution
2.1.70	Pipe
2.1.71	Portable Filename Character Set
2.1.72	Portable Pathname
2.1.73	Privilege
2.1.74	Process
2.1.75	Process Group
2.1.76	Process Group ID
2.1.77	Process Group Leader
2.1.78	Process Group Lifetime
2.1.79	Process ID
2.1.80	Process Lifetime
2.1.81	Read-only File System
2.1.82	Real Group ID
2.1.83	Real User ID
2.1.84	Regular File
2.1.85	Relative Pathname
2.1.86	Root Directory
2.1.87	Saved set-group-ID
2.1.88	Saved set-user-ID
2.1.89	SCCS
2.1.90	Session
2.1.91	Session Leader
2.1.92	Session Lifetime
2.1.93	Signal
2.1.94	Slash
2.1.95	Supplementary Group ID
2.1.96	System
2.1.97	System Process
2.1.98	Terminal (or Terminal Device)
2.1.99	User ID
2.1.100	Working Directory (or Current Working Directory)
2.1.101	Zombie Process
2.1.102	[n, m] and [n, m)
2.2	SHELL METANOTATION
2.2.1	Eight Bit Transparent Utilities
2.3	DIRECTORY STRUCTURE
2.4	ENVIRONMENT VARIABLES
2.5	DEVICES
2.5.1	/dev/null
2.5.2	/dev/tty
2.6	OUTPUT DEVICES

2.7	SHELL BUILT-INS
Chapter 3	COMMANDS AND UTILITIES
	<i>admin</i>
	<i>ar</i>
	<i>at</i>
	<i>awk</i>
	<i>banner</i>
	<i>basename</i>
	<i>cal</i>
	<i>calendar</i>
	<i>cat</i>
	<i>cc</i>
	<i>cd</i>
	<i>cflow</i>
	<i>chmod</i>
	<i>chown</i>
	<i>chroot</i>
	<i>cmp</i>
	<i>col</i>
	<i>comm</i>
	<i>cp</i>
	<i>cpio</i>
	<i>crontab</i>
	<i>csplit</i>
	<i>cu</i>
	<i>cut</i>
	<i>cxref</i>
	<i>date</i>
	<i>dd</i>
	<i>delta</i>
	<i>df</i>
	<i>diff</i>
	<i>dircmp</i>
	<i>dis</i>
	<i>du</i>
	<i>echo</i>
	<i>ed</i>
	<i>egrep</i>
	<i>env</i>
	<i>ex</i>
	<i>expr</i>
	<i>find</i>
	<i>gencat</i>
	<i>get</i>
	<i>grep</i>
	<i>iconv</i>

Contents

id
join
kill
lex
line
lint
logname
lp
lpstat
ls
m4
mail
mailx
make
mesg
mkdir
mkfifo
newgrp
nl
nm
nohup
od
pack
paste
pg
pr
prs
ps
pwd
rm
rmdel
sact
sdb
sed
sh
sleep
sort
spell
split
strip
stty
sum
tabs
tail
tar
tee
test

time
touch
tr
true
tsort
tty
umask
uname
unget
uniq
uucp
uustat
uuto
uux
val
vi
wait
wall
wc
what
who
write
xargs
yacc

Appendix A

WITHDRAWN INTERFACES

as
cpp
file
ld
lorder
mknod
news
passwd
prof
shl
size
su

Preface

X/Open represents a major breakthrough in the world of standards for the information technology industry. A large number of the world's major information system suppliers have come together to encourage applications portability resulting in tangible benefits for computer users, independent software houses and for the suppliers themselves.

The X/Open Group's principal aim is to increase the volume of applications available and to maximise the return on investments in software development made by users and independent software vendors.

This is achieved by ensuring the portability of application programs at the source code level. Through this portability, users can mix and match computer systems and applications software from many suppliers, and thus investment in applications software is protected into the future.

In order to provide such portability, the X/Open Group has defined a **Common Applications Environment** which is built on the interfaces to the UNIX operating system, and covers other aspects required of a comprehensive applications interface.

The X/Open Portability Guide contains an evolving portfolio of practical standards for application portability. All of the members of X/Open guarantee to support the standards defined, leading to:

- Growing portability
- No dependence on a single source – freedom of choice
- Increased application software selection
- More security in software investments
- International support for the **Common Applications Environment**

X/Open is not a standards-setting organisation; it is a joint initiative by members of the business community to integrate evolving standards into a common, beneficial and continuing strategy.

Trademarks

UNIX[®] is a registered trademark of AT&T in the USA and other countries.

X/Open[™] is a licensed trademark of the X/Open Group Members.

AT&T[®] is a registered trademark of American Telephone & Telegraph in the USA and other countries.

Acknowledgements

X/Open gratefully acknowledges:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The **/usr/group** Standards Committee, whose Standard contributed to the Group's work.
- The **/usr/group** Technical Subcommittee Internationalisation for work on Internationalised Regular Expressions.

Referenced Documents

The following documents are referenced in this guide:

- System V Interface Definition (Spring 1985 - Issue 1)
- System V Interface Definition (Spring 1986 - Issue 2)
- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2)
- UNIX System V - Release 2.0 Programming Guide (April 1984 - Issue 2)
- ANSI X3.159, Programming Language C
- 1984 /usr/group Standard
- IEEE Std. 754-1985
- IEEE P1003.1 Trial Use Standard (April 1986)
- IEEE Std. 1003.1-1988
- Draft IEEE P1003.2
- ISO 6937:1983
- ISO 8859-1:1987

Introduction

1.1 OVERVIEW

1.1.1 Rationale

The X/Open System Interface (XSI) specification is separated into **Volume 1, XSI Commands and Utilities** and **Volume 2, XSI System Interfaces and Headers**.

This volume defines utilities which are accessed using commands given to command interpreters. The system interface and headers and the utilities are jointly known as services.

The XSI defines the interfaces and run-time behaviour provided by the services to application programs. No particular restrictions are imposed on the way in which the services are implemented.

The utilities are defined in terms of their interface as seen from the *sh* command interpreter. Alternative interfaces are available to application programs through one of the *exec* functions, and the *popen()* and *system()* interfaces.

1.1.2 Important Note

Little material in this volume has been updated since Issue 2. The following should be considered:

- As in XPG Issue 2, the current descriptions of the utilities are incomplete and lack precision.
- Certain utilities in this volume rely on the fundamental behaviour of XSI interfaces defined in Volume 2. The descriptions of these utilities have not been updated to reflect all changes to Volume 2. Application writers should be aware that utilities may exhibit behaviour not described in this volume but which is in line with functionality described in Volume 2.
- There are many inconsistencies in the use of terminology in this volume. Terms with different meanings are used interchangeably. For instance, the terms "character" and "byte" or the terms "space" and "blank".

Therefore, this volume in its current form is useful as a guide to portability, but it is not possible to precisely define or test conformance to it.

The **IEEE P1003.2 Shell and Application Utilities Interface for Computer Operating System Environments** Working Group is currently redefining most of the utilities in this volume. This volume will be aligned with **IEEE Std. 1003.2-19xx** after **IEEE Std. 1003.2-19xx** attains full use status. During this alignment process the problems noted above will be addressed.

1.2 STATUS OF INTERFACES

1.2.1 Mandatory

The majority of the utilities are mandatory; they must be present in all X/Open compliant systems and they must conform to the published definition. The commands and utilities are all to be found under a single heading. They are split functionally into those that are intended to provide an applications interface (referred to as *standard utilities*) and those that are intended to be used only by development programmers or during the porting of applications to an X/Open Compliant System (referred to as *development utilities*).

1.2.2 Optional

A small number of utilities are optional. The presence of these is not mandatory, although if they are present they must conform to the definition. The list below shows those that are optional.

Utilities marked in this issue as optional are:

Optional Utilities	
<i>dis</i>	<i>newgrp</i>
<i>mailx</i>	<i>sdb</i>

In addition, there are a number of utilities that may not be supportable on X/Open compliant systems (see **Section 1.2.6, Portability**).

1.2.3 Development

The development utilities might not be present in all X/Open compliant systems; in designated (DEVELOPMENT) systems all of the development utilities must be present and must conform to the published definition.

Utilities marked in this issue as development are:

Development Utilities			
<i>admin</i>	<i>dis</i>	<i>m4</i>	<i>rmdel</i>
<i>cflow</i>	<i>get</i>	<i>make</i>	<i>sact</i>
<i>cxref</i>	<i>lex</i>	<i>nm</i>	<i>sdb</i>
<i>delta</i>	<i>lint</i>	<i>prs</i>	<i>strip</i>

1.2.4 New Interfaces

A few interfaces are new in this issue of the Guide. These are:

New Interfaces	
<i>iconv</i>	<i>mkfifo</i>
<i>gencat</i>	

1.2.5 Withdrawn

Some of the interfaces in this issue have been marked as withdrawn. Various factors may have contributed to the decision to withdraw an interface. Wherever possible, the reasons for withdrawal of an interface are documented on the relevant pages.

If a migration path exists, advice is given to application developers regarding alternative means of obtaining similar functionality. This information may be found in the **APPLICATION USAGE** sections on the relevant pages.

Withdrawn interfaces may still exist on conformant implementations. However, they will be dropped from the next issue of the Guide. Application writers should not use functionality marked as withdrawn.

Interfaces marked in this issue as withdrawn are:

Withdrawn	
<i>as</i>	<i>news</i>
<i>cpp</i>	<i>passwd</i>
<i>file</i>	<i>prof</i>
<i>ld</i>	<i>shl</i>
<i>lorder</i>	<i>size</i>
<i>mknod</i>	<i>su</i>

1.2.6 Portability

Commands and utilities offer functionality and interfaces that are substantially more complex than that of the system interface routines. They are in many cases much harder to define and it is therefore more difficult to guarantee portability of interfaces and consistent behaviour between systems.

The definition of standards for commands and utilities is an ongoing activity in the IEEE P1003.2 Working Group. X/Open is participating in this work.

The current definition is a valuable first step, but additional work will be needed to evolve towards a complete guide. For example:

- The number of options defined for many of the utilities is excessive and includes functionality which is rarely used or is implementation specific.
- There are too many different ways of achieving the same results.
- Many of the current descriptions were written to record the observed behaviour of already existing utilities and the level of precision is inadequate for use as a definitive guide.

Rectification of this is an enormous task and the current X/Open definition is of necessity based on the available documentation, but incorporates annotation to highlight potential portability problems resulting from the points listed above.

X/Open is working on a substantially improved definition of commands, with the number of options reduced to those in common use and with a higher level of specification.

This improvement process is being carried out in close consultation with the various user organisations and standards bodies, such as IEEE and ISO, to ensure that the result is a single standard definition of operating system commands and utilities. The new definition will align with that in IEEE P1003.2 when it is published.

Readers of this Guide are invited to participate directly in the consultative process to ensure that the evolving standard matches the requirements of existing and future applications. The preface to this volume of the Guide includes details of how to submit comments directly to the X/Open Company.

Wherever reduced functionality is highlighted, to warn that reduced portability exists, the reason is shown by a code in the margin which refers to the list given below. In all cases an application wishing to achieve maximum portability should avoid such functionality.

Unless the primary task of a utility is to produce textual material on its standard output, application developers should not rely on the format or content of any such material which may be produced. Where the primary task *is* to provide such material, but the output format is incompletely specified, the description is marked. Applications developers are warned not to expect that the output of such a utility on one system will be any guide to its behaviour on another system.

It is expected that the output from development utilities will be read by an expert user who can interpret it, and no warning is given where their output format is incompletely specified or has a language or cultural dependency.

The warning codes and their meanings are as follows:

- IN Internationalised functionality.
This functionality requires internationalisation support from the implementation, and might not be present on systems that do not support the internationalised utility environment described in Volume 3, XSI

Internationalisation. Systems that support internationalised utilities provide all the marked facilities, i.e., this functionality is collectively optional.

- LA Language or cultural dependency.
The output produced by the utility is in a language or format which may not be understood by all users.
- MV Marginal value.
The functionality described is of marginal value and may be removed in future editions of the Guide.
- OF Output format incompletely specified.
The format of the output produced by the utility is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.
- OP Dependent on optional service in XSI.
Typical implementations depend on an optional service and the functionality affected may not be present if the optional service is not supported.
- PI The behaviour cannot be guaranteed to be consistent.
It is not possible to guarantee that the utility behaves in the same way on all X/Open compliant systems. This is the case if it provides functionality which is system-defined or system-specific. Options which are used to *select* alternative forms of system-specific behaviour are not marked, as it is clear from their descriptions that their use is inherently non-portable.
- UN Possibly unsupportable feature.
It may not be possible to implement the required functionality (as defined) on all X/Open compliant systems and the functionality may not be present. This may, for example, be the case where the X/Open compliant system is hosted and the underlying system provides the service in an alternative way.

1.3 RELATIONSHIP TO OTHER STANDARDS

Little material has changed in this volume between Issue 2 and Issue 3, see **Section 1.1.2, Important Note**.

The **IEEE P1003.2** Working group are currently working on a standard for Commands and Utilities. X/Open is working closely with the **IEEE P1003.2** Working group in an effort to complete the standardisation process as soon as possible.

Once the standardisation work is complete, the entries in this volume will be aligned with the standard. It is anticipated that this will involve the addition of much greater detail, some new functionality, and a small number of changes.

1.4 FORMAT OF ENTRIES

The entries in each chapter are based on a common format, not all of whose parts always appear.

Each entry describes one or more of the commands and utilities. Each page is identified at the top outer corner by the entry name.

NAME

gives the name or names of the services described by the entry and briefly states their purpose.

SYNOPSIS

summarises the use of the service being described.

DESCRIPTION

defines the functionality and behaviour of the service.

EXIT STATUS

describes the values the utility may return to the invoking program and the conditions that cause the value to be returned.

ERRORS

describes conditions that may give rise to an error.

EXAMPLE(S)

gives example(s) of usage, where appropriate.

APPLICATION USAGE gives additional information about the way that the utility should be used.

FUTURE DIRECTIONS

should be used as a guide to current thinking; there is not necessarily a commitment to implement all of these future directions in their entirety.

SEE ALSO

lists related entries.

CHANGE HISTORY

shows the derivation of the description used by the XSI, and lists the functional differences between Issue 2 and Issue 3.

The formal description consists only of the **NAME**, **SYNOPSIS**, **DESCRIPTION**, **ERRORS** and **EXIT STATUS** parts.

1.4.1 Typographical Conventions

The following typographical conventions are used:

- **Boldface** strings are literals and are to be typed just as they appear.
- *Italic* strings usually represent substitutable argument prototypes and the names of entries found elsewhere.
- Names in upper-case surrounded by braces, e.g., {CONST}, represent system-dependent constants.
- The notation *name*() indicates a function, which is supplied as part of the applications development system, see **Volume 2, XSI System Interfaces and Headers, Chapter 3**. Names without parentheses may be either variable names, also in **Volume 2, XSI System Interfaces and Headers, Chapter 3**, or utilities in **Chapter 3** of this Volume.
- The notation <file.h> indicates a header, which is supplied as part of the applications development system, see **Volume 2, XSI System Interfaces and Headers, Chapter 4, Headers** and **Volume 4, C Language, Section 2.11.2, File Inclusion**.
- Square brackets [] around an argument prototype indicate that the argument is optional. When an argument prototype is given as *name* or *file*, it always refers to a pathname.
- Ellipses, "...", are used to show that the previous argument may be repeated.

Definitions

2.1 GLOSSARY

Many special terms are used in the interface definitions. The descriptions of these terms follow.

2.1.1 Absolute Pathname

See **Section 2.1.69, Pathname Resolution**.

2.1.2 Access Mode

The particular form of access permitted to a file.

2.1.3 Address Space

The memory locations that can be referenced by a process.

2.1.4 Appropriate Privileges

An implementation-defined means of associating privileges with a process with regard to the function calls and function call options defined in this Guide that need special privileges. There may be zero or more such means.

2.1.5 Background Process Group

Any process group that is a member of a session that has established a connection with a controlling terminal that is not in the foreground process group.

2.1.6 Block Special File

A file that refers to a device. A block special file is normally distinguished from a character special file by providing access to the device in a manner such that the hardware characteristics of the device are not visible.

2.1.7 C Standard

The abbreviated name for **Draft ANSI X3.159, Programming Language C**.

2.1.8 Character

A sequence of one or more bytes representing a single graphic symbol.

2.1.9 Character Sets

For a full definition refer to **Volume 3, XSI Internationalisation, Chapter 7, C Program Locale**.

2.1.10 Character Special File

A file that refers to a device. One type of character special file is a terminal device file whose access is defined in **Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**.

2.1.11 Child Process

See **Section 2.1.74, Process**.

2.1.12 Clock Tick

The number of intervals per second, defined by {CLK_TCK}, used to express the value in type `clock_t`.

2.1.13 Command Interpreter

It is possible for applications to invoke utilities through a number of interfaces, which are collectively considered to act as command interpreters. The most obvious of these are *sh* (**Volume 1, XSI Commands and Utilities**) and *system()*, although *popen()* (**Volume 2, XSI System Interfaces and Headers**) and the various forms of *exec* may also be considered to behave as interpreters.

2.1.14 Controlling Process

The session leader that established the connection to the controlling terminal. If the terminal ceases to be a controlling terminal for this session, the session leader will cease to be the controlling process.

2.1.15 Controlling Terminal

A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) cause signals to be sent to all processes in the process group associated with the controlling terminal.

2.1.16 Current Working Directory

See **Section 2.1.100, Working Directory**.

2.1.17 Device

A computer peripheral or an object that appears to the application as such.

2.1.18 Device ID

A non-negative integer used to identify a device.

2.1.19 Directory

A file that contains directory entries. No two directory entries in the same directory have the same name.

2.1.20 Directory Entry

An object that associates a filename with a file. Several directory entries can associate names with the same file.

2.1.21 Directory Stream

A per process unique identifier used to reference an open directory.

2.1.22 Dot

The filename consisting of a single dot character, ".". See **Section 2.1.69, Pathname Resolution**.

2.1.23 Dot-dot

The filename consisting of two dot characters, "..". See **Section 2.1.69, Pathname Resolution**.

2.1.24 Effective Group ID

An attribute of a process which is used to determine various permissions, including file access permissions, see **Section 2.1.49, Group ID**. This value is subject to change during the process lifetime, as described in *setgid()* and *exec* (**Volume 2, XSI System Interfaces and Headers**).

2.1.25 Effective User ID

An attribute of a process which is used to determine various permissions, including file access permissions, see **Section 2.1.99, User ID**. This value is subject to change during the process lifetime, as described in *setuid()* and *exec* (**Volume 2, XSI System Interfaces and Headers**).

2.1.26 Empty Directory

A directory that contains, at most, directory entries for dot and dot-dot.

2.1.27 Empty String

The empty string (or null string) is a character array whose first element is a null character.

2.1.28 Epoch

The time 0 hours, 0 minutes, 0 seconds, on January 1, 1970 Coordinated Universal Time.

2.1.29 Extended Security Controls

The access control (see **Section 2.1.33, File Access Permissions**) and privilege (see **Section 2.1.4, Appropriate Privileges**) have been defined to allow implementation-defined extended security controls. These permit an implementation to provide security mechanisms to implement different security policies than are described in this Guide. These mechanisms will not alter or override the defined semantics of any of the functions in this Guide.

2.1.30 Feature Test Macro

A macro used to determine whether a particular set of features will be included from a header. This is described in **Volume 2, XSI System Interfaces and Headers, Section 1.7, The Compilation Environment**.

2.1.31 FIFO Special Files (or FIFO)

A type of file. Data written to a FIFO special file are read on a first-in-first-out basis. Other characteristics of FIFOs are described under *lseek()*, *open()*, *read()* and *write()* (**Volume 2, XSI System Interfaces and Headers**).

2.1.32 File

An object that can be written to and/or read from. A file has certain attributes, including file access permissions and type. File types include regular file, character special file, block special file, FIFO special file and directory. Other types of files may be defined by the implementation.

2.1.33 File Access Permissions

The standard file access control mechanism uses the file permission bits, as described below. These bits are set at the time of file creation by *open()*, *creat()*, *mkdir()* and *mkfifo()* and are changed by *chmod()*. These bits are read by *stat()* or *fstat()* (**Volume 2, XSI System Interfaces and Headers**).

Implementations may provide *additional* or *alternate* file access mechanisms, or both. An additional file access control mechanism will only further restrict the access permissions defined by the file permission bits. An alternate file access control mechanism will:

- Specify file permission bits for the file owner class, the file group class and the file other class of that file, corresponding to the access permissions to be returned by *stat()* or *fstat()*.
- Be enabled only by explicit user action, on a per-file basis by the file owner or by a user with appropriate privilege.
- Be disabled after the file permission bits are changed with *chmod()*. The disabling of the alternate mechanism need not disable any additional mechanisms defined by the implementation.

Whenever a process requests file access permission for read, write or execute/search, if no additional mechanism denies access, access is determined as follows:

- If a process has appropriate privilege:
 - If read, write or directory search permission is requested, access is granted.
 - If execute permission is requested, access is granted if execute permission is granted to at least one user by the file permission bits, or by an alternate access control mechanism; otherwise access is denied.
- Otherwise:
 - The file permission bits of a file contain read, write and execute/search permissions for the file owner class, file group class and file other class. Access is granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise access is denied.

2.1.34 File Description

See Section 2.1.61, Open File Description.

2.1.35 File Descriptor

A file descriptor is a per-process unique integer used to identify an open file for the purpose of file access. The value of a file descriptor is from 0 to (OPEN_MAX)-1. A process may have no more than (OPEN_MAX) file descriptors open simultaneously. See Section 2.1.61, Open File Description.

2.1.36 File Group Class

A process is in the file group class of a file if the process is not in the file owner class and if the effective group ID or one of the supplementary group IDs of the process matches the group ID associated with the file. Other members of the class may be implementation-defined.

2.1.37 File Hierarchy

Files in the system are organised in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any type of file. Because multiple directory entries may refer to the same file, the hierarchy is properly described as a *directed graph*.

2.1.38 File Mode

The file mode contains the file permission bits and other characteristics of the file, as described in <sys/stat.h> (Volume 2, XSI System Interfaces and Headers).

2.1.39 Filename

Names consisting of 1 to {NAME_MAX} bytes may be used to name a file. The characters constituting the name may be selected from the set of all character values excluding the slash and null characters. The filenames dot and dot-dot have special meaning; see **Section 2.1.69, Pathname Resolution**. A filename is sometimes referred to as a "pathname component".

Filenames should be constructed from the Portable Filename Character Set because the use of other characters can be confusing or ambiguous in certain contexts.

2.1.40 File Offset

This specifies the byte position in the file where the next I/O operation begins. Each open file description associated with a regular file, block special file or directory has a file offset. A character special file that does not refer to a terminal device may have a file offset. There is no file offset specified for a pipe or FIFO.

2.1.41 File Other Class

A process is in the file other class of a file if the process is not in the file owner class or file group class.

2.1.42 File Owner Class

A process is in the file owner class of a file if the effective user ID of the process matches the user ID of the file.

2.1.43 File Permission Bits

These are used, along with other information, to determine whether a process has read, write or execute/search permission to a file. The bits are divided into three parts: owner, group and other. Each part is used with the corresponding file class of processes. These bits are contained in the file mode, as described in <sys/stat.h> (**Volume 2, XSI System Interfaces and Headers**). The detailed usage of the file permission bits in access decisions is described in file access permissions.

2.1.44 File Serial Number

A per-file system unique identifier for a file.

2.1.45 File System

A collection of files and certain of their attributes. It provides a name space for file serial numbers referring to those files.

2.1.46 File Times Update

Each file has three associated time values that are updated when file data has been accessed, file data has been modified, or file status has been changed, respectively. These values are returned in the file characteristics structure, as described in <sys/stat.h> (**Volume 2, XSI System Interfaces and Headers**).

For each function in this Guide that reads or writes file data or changes the file status, the appropriate time-related fields are noted as “marked for update”. At the next time an update occurs, any marked fields are set to the current time and the update marks cleared. Two such update points are when the file is no longer open by any process and when *stat()* or *fstat()* (**Volume 2, XSI System Interfaces and Headers**) are performed on the file. Additional update points are unspecified. Updates are not done for files on read-only file systems.

2.1.47 Foreground Process Group

Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. The foreground process group has certain privileges when accessing its controlling terminal that are denied to background process groups. This is described in **Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**.

2.1.48 Foreground Process Group ID

The process group ID of the foreground process group.

2.1.49 Group ID

Each system user is a member of at least one group. A group is identified by a group ID, a non-negative integer that can be contained in a *gid_t*. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID or one of the *optional* supplementary group IDs or as a saved set-group-ID.

2.1.50 Job Control

This allows users to selectively stop (suspend) the execution of processes and to continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. XSI compliant systems may optionally support job control facilities; the presence of this option is indicated to the application at compile time or run time by the definition of the `{_POSIX_JOB_CONTROL}` symbol.

2.1.51 Link

See **Section 2.1.20, Directory Entry**.

2.1.52 Link Count

The link count of a file is the number of directory entries that refer to that file.

2.1.53 Message Catalogue

A file or storage area containing program messages, command prompts and responses to prompts for a particular native language, territory and codeset.

2.1.54 Message Catalogue Descriptor

A per process unique value of type `nl_catd` used to identify an open message catalogue.

2.1.55 Mode

The mode of a file is a collection of attributes that specifies the file's type and its access permissions.

2.1.56 NaN (Not a Number)

A value that can be stored in a real type but which is not a valid floating point number.

2.1.57 Null Character

A byte with all bits set to 0.

2.1.58 Null Pointer

This is the value that is obtained by casting 0 into a pointer e.g., `(char *) 0`. The C-language guarantees that this value will not match that of any legitimate pointer, so it is used by many functions that return pointers to indicate an error.

2.1.59 Null String

See Section 2.1.27, Empty String.

2.1.60 Open File

A file that is currently associated with a file descriptor.

2.1.61 Open File Description

This records how a process or group of processes are accessing a file. Each file descriptor refers to exactly one open file description, but an open file description can be referred to by more than one file descriptor. The file offset, file status and file access modes are attributes of an open file description.

2.1.62 Orphaned Process Group

A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session.

2.1.63 Parent Directory

The directory containing a directory entry for the file under discussion. This concept does not apply to dot and dot-dot.

2.1.64 Parent Process

See Section 2.1.74, Process.

2.1.65 Parent Process ID

A new process is created by a currently active process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined process.

2.1.66 Path Prefix

A pathname, with optional ending slash, that refers to a directory.

2.1.67 Pathname

A string that is used to identify a file. It consists of, at most, {PATH_MAX} bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple consecutive slashes are considered the same as one slash. A pathname that begins with two consecutive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes is treated as a single slash. The interpretation of the pathname is described in **Section 2.1.69, Pathname Resolution**.

2.1.68 Pathname Component

See **Section 2.1.39, Filename**.

2.1.69 Pathname Resolution

This is performed for a process to resolve a pathname to a particular file in a file hierarchy. There may be multiple pathnames that resolve to the same file.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment "a/b", file "b" is located in directory "a"). Pathname resolution fails if this cannot be accomplished. If the pathname begins with a slash, the predecessor of the first filename in the pathname is taken to be the root directory of the process (such pathnames are referred to as "absolute pathnames"). If the pathname does not begin with a slash, the predecessor of the first filename of the pathname is taken to be current working directory of the process (such pathnames are referred to as "relative pathnames").

The interpretation of a pathname component is dependent on the values of {NAME_MAX} and {_POSIX_NO_TRUNC} associated with the path prefix of that component. If any pathname component is longer than {NAME_MAX}, and {_POSIX_NO_TRUNC} is in effect for the path prefix of that component (see *pathconf()*, in **Volume 2, XSI System Interfaces and Headers**), the implementation will consider this an error condition. Otherwise, the implementation will use the first {NAME_MAX} bytes of the pathname component.

The special filename dot, ".", refers to the directory specified by its predecessor. The special filename dot-dot, "..", refers to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself.

A pathname consisting of a single slash resolves to the root directory of the process. A null pathname is invalid.

2.1.70 Pipe

An object accessed by one of the pair of file descriptors created by the *pipe()* function (**Volume 2, XSI System Interfaces and Headers**). Once created, the file descriptors can be used to manipulate it and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy.

2.1.71 Portable Filename Character Set

For a filename to be portable across conforming implementations of **IEEE Std. 1003.1-1988** it must consist only of the following characters:

```

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

```

The last three characters are the dot, underscore and hyphen characters, respectively.

The hyphen should not be used as the first character of the portable filename. Upper- and lower-case letters retain their unique identities between conforming implementations.

For a filename to be portable across implementations conforming to the **X/Open Portability Guide Issue 3**, it may consist of any of the **ISO 8859-1:1987** characters:

```

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9
! " # $ % & ' ( ) * + , - . : ; < = > ? @ [ \ ] ^ _ { | } ~
¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ

```

2.1.72 Portable Pathname

For a pathname to be portable across XSI compliant systems, it should consist of at most [PATH_MAX] bytes, including the terminating null character. It should be a pathname consisting of an optional beginning slash, followed by zero or more portable filenames separated by slashes.

2.1.73 Privilege

See Section 2.1.4, **Appropriate Privileges**.

2.1.74 Process

An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the *fork()* function (**Volume 2, XSI System Interfaces and Headers**). The process that issues *fork()* is known as the parent process, and the new process created by the *fork()* as the child process.

2.1.75 Process Group

Each process in the system is a member of a process group which is identified by a process group ID. This grouping permits the signalling of related groups of processes. A newly created process joins the process group of its creator.

2.1.76 Process Group ID

Each process group in the system is uniquely identified during its lifetime by a positive integer that can be contained in a *pid_t* called a process group ID. A process group ID may not be reused by the system until the process group lifetime ends.

2.1.77 Process Group Leader

A process whose process ID is the same as its process group ID.

2.1.78 Process Group Lifetime

A period of time that begins when a process group is created and ends when the last remaining process in the group leaves the group, either due to process termination or calling the *setsid()* or *setpgid()* functions (**Volume 2, XSI System Interfaces and Headers**).

2.1.79 Process ID

Each active process in the system is uniquely identified during its lifetime by a positive integer that can be contained in a *pid_t* called a process ID. A process ID may not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID may not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1.

2.1.80 Process Lifetime

After a process is created with the *fork()* function (**Volume 2, XSI System Interfaces and Headers**), it is considered active. Its thread of control and address space exist until it terminates. It then enters an inactive state where certain resources may be returned to the system, although some resources, such as the process ID, are still in use. When another process executes a *wait()* or *waitpid()*

function (again, described in **Volume 2**) for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends.

2.1.81 Read-only File System

A file system that has an implementation-defined characteristic restricting modifications.

2.1.82 Real Group ID

The attribute of a process that, at the time of process creation, identifies the group of the user who created the process. See **Section 2.1.49, Group ID**. This value is subject to change during the process lifetime, as described in *setgid()* (**Volume 2, XSI System Interfaces and Headers**).

2.1.83 Real User ID

The attribute of a process that, at the time of process creation, identifies the user who created the process. See **Section 2.1.99, User ID**. This value is subject to change during the process lifetime, as described in *setuid()* (**Volume 2, XSI System Interfaces and Headers**).

2.1.84 Regular File

A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system.

2.1.85 Relative Pathname

See **Section 2.1.69, Pathname Resolution**.

2.1.86 Root Directory

A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash.

2.1.87 Saved set-group-ID

This is an attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, as described in *setgid()* and *exec* (**Volume 2, XSI System Interfaces and Headers**).

2.1.88 Saved set-user-ID

This is an attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, as described in *setuid()* and *exec* (**Volume 2, XSI System Interfaces and Headers**).

2.1.89 SCCS

A set of utilities providing facilities for the administration of source code files.

2.1.90 Session

Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly-created process joins the session of its creator. A process can alter its session membership (see `setsid()`, in **Volume 2, XSI System Interfaces and Headers**). Implementations which support `setpgid()` can have multiple process groups in the same session.

2.1.91 Session Leader

This is a process that has created a session (see `setsid()`, in **Volume 2, XSI System Interfaces and Headers**).

2.1.92 Session Lifetime

The period between when a session is created and the end of the lifetime of all the process groups that remain as members of the session.

2.1.93 Signal

A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term "signal" is also used to refer to the event itself.

2.1.94 Slash

The term "slash" is used to represent the literal character `"/"`. This character is also known as a "solidus" in **ISO 8859-1:1987**.

2.1.95 Supplementary Group ID

A process has up to `{NGROUPS_MAX}` supplementary group IDs used in determining file access permissions, in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. Whether a process's effective group ID is included in or omitted from its list of supplementary group IDs is unspecified.

2.1.96 System

The term "system" is used in this Guide to refer to an implementation of the XSI.

2.1.97 System Process

An object, other than a process executing an application, that is defined by the system and has a process ID.

2.1.98 Terminal (or Terminal Device)

A character special file that obeys the specifications of the general terminal interface as described in **Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**.

2.1.99 User ID

Each system user is identified by a non-negative integer that can be contained in a `uid_t`. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID or a saved set-user-ID.

2.1.100 Working Directory (or Current Working Directory)

A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash.

2.1.101 Zombie Process

An inactive process which will be deleted at some later time when its parent process executes `wait()` or `waitpid()` (**Volume 2, XSI System Interfaces and Headers**).

2.1.102 [n, m] and [n, m)

This notation denotes a mathematical range. The brackets "[" and "]" include the limit; the parentheses "(" and ")" exclude the limit, i.e., if x is in $[0, 1]$, it can be from 0 to 1 inclusive, but if x is in $[0, 1)$, it can be from 0 up to but not including 1.

2.2 SHELL METANOTATION

The shell metanotation characters are similar to regular expressions. The important difference is that the shell metanotation characters are used to match filenames.

The following metacharacters are defined:

- * Matches any string, including the empty string.
- ? Matches any single character.
- [...] Matches any one of the enclosed characters. Characters in the list may be specified as collating elements or symbols, range expressions, character classes or equivalence classes. A pair of characters separated by "-" matches any symbol between the pair (inclusive), as defined by collating sequence information in the shell's locale. If the first character following the opening "[" is a "!", any character not enclosed is matched.

These characters are used by the following commands and utilities:

Utilities using Shell Metanotation	
<i>cpio</i>	<i>find</i>
<i>sh</i>	<i>uux</i>

2.2.1 Eight Bit Transparent Utilities

The following standard utilities are identified as the minimum set of commands that will provide 8-bit transparency on all conforming systems:

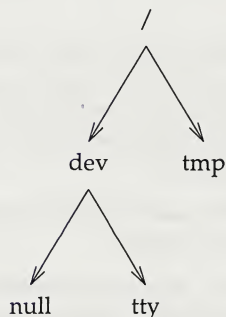
Eight Bit Transparent Utilities			
<i>ar</i>	<i>expr</i>	<i>pcat</i>	<i>true</i>
<i>awk</i>	<i>false</i>	<i>pg</i>	<i>tty</i>
<i>cancel</i>	<i>fgrep</i>	<i>pr</i>	<i>umask</i>
<i>cat</i>	<i>find</i>	<i>ps</i>	<i>uname</i>
<i>cc</i>	<i>gencat</i>	<i>pwd</i>	<i>uniq</i>
<i>cd</i>	<i>grep</i>	<i>red</i>	<i>unpack</i>
<i>chgrp</i>	<i>iconv</i>	<i>rm</i>	<i>uucp</i>
<i>chmod</i>	<i>kill</i>	<i>rmdir</i>	<i>uulog</i>
<i>chown</i>	<i>lex</i>	<i>sed</i>	<i>uuname</i>
<i>cmp</i>	<i>ln</i>	<i>sh</i>	<i>uupick</i>
<i>comm</i>	<i>lp</i>	<i>sleep</i>	<i>uustat</i>
<i>cp</i>	<i>lpstat</i>	<i>sort</i>	<i>uto</i>
<i>cpio</i>	<i>ls</i>	<i>stty</i>	<i>uux</i>
<i>date</i>	<i>mail</i>	<i>tail</i>	<i>wait</i>
<i>diff</i>	<i>mailx</i>	<i>tar</i>	<i>wc</i>
<i>echo</i>	<i>mkdir</i>	<i>tee</i>	<i>who</i>
<i>ed</i>	<i>mv</i>	<i>test</i>	<i>yacc</i>
<i>egrep</i>	<i>pack</i>	<i>tr</i>	

In future versions of XPG, all utilities will be at least 8-bit clear, and will eventually move to internationalisation.

Note that the provision of message catalogues for these commands is not covered by the XSI.

2.3 DIRECTORY STRUCTURE

Below is a diagram of the minimal directory tree structure present on any X/Open compliant system.



The following guidelines apply:

/dev contains special files (I/O devices). Those that are always present in X/Open compliant systems are defined in **Volume 2, XSI System Interfaces and Headers, Section 2.5, Devices**.

Applications wishing to create temporary files should do so in the directory named by the **TMPDIR** environment variable (see **Volume 2, XSI System Interfaces and Headers, Section 2.2, Environment Variables**). If **TMPDIR** does not exist, or is null, applications may try the **/tmp** directory instead.

If the system is restarted after being halted for any reason, applications cannot rely on the contents of **/tmp** remaining undisturbed. It is common for it to be emptied by the system restart procedures.

2.4 ENVIRONMENT VARIABLES

An array of strings called the environment is made available when a process begins. The array is pointed to by the external variable *environ* which is defined as

```
extern char **environ;
```

These strings have the form "*name=value*"; *names* do not contain the character "*=*". There is no meaning associated with the order of strings in the environment. If more than one string in a process's environment has the same *name*, the consequences are undefined. The following names may be defined and have the indicated meaning if they are defined:

- | | |
|-------------|--|
| HOME | Name of the user's initial working directory, from the user database, see <code><pwd.h></code> . |
| LANG | Identifies the user's requirements for native language, local customs and coded character set, as an ASCII character string, in the form |

```
LANG=language[_territory[.codeset]]
```

where standard settings of *language*, *territory* and *codeset* are implementation defined (see `<lang.h>`).

Specific language operation is initiated at run-time by calling the *setlocale()*, function. Normally, the user's language requirements, as specified by the setting of *LANG*, are bound to a program's locale by calling *setlocale()* as follows:

```
setlocale(LC_ALL, "");
```

On X/Open systems, this form of a *setlocale()* call is defined to initialise the program locale from the associated environment variables. *LC_ALL* names the program's entire locale and *LANG* provides the necessary defaults if any of the category-specific variables is not set or is set to the empty string (as described in Volume 3, XSI Internationalisation, Section 4.2.5, Setting a Specific Category to the Implementation-defined Default).

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME

Contain the user's requirements for language, territory and codeset with respect to character collation, character classification and conversion, currency symbol and monetary value format, numeric data presentation and time formats respectively. If any of these are not defined in the current environment, *LANG* provides the necessary defaults.

Category	Summary
LC_COLLATE	affects the behaviour of regular expressions and string collation functions in <i>strcoll()</i> and <i>strxfrm()</i> .
LC_CTYPE	affects the behaviour of regular expressions and the character handling functions in <i>tolower()</i> , <i>toupper()</i> , <i>isalpha()</i> , etc.
LC_MONETARY	Contains the user's requirements for language, territory and codeset with respect to monetary values.
LC_NUMERIC	affects the radix character for the formatted input/output functions in <i>scanf()</i> and <i>printf()</i> , and the string conversion function in <i>strtod()</i> .
LC_TIME	affects the behaviour of the time functions in <i>strftime()</i> .

The behaviour of the language information function defined in *nl_langinfo()* is also affected by the setting of these environment variables (see *<langinfo.h>*).

LC_COLLATE, *LC_CTYPE*, *LC_MONETARY*, *LC_NUMERIC* and *LC_TIME* are defined to accept an additional field "@modifier", which allows the user to select a specific instance of localisation data within a single category (e.g., for selecting the dictionary as opposed to the character ordering of data). The syntax for these environment variables is thus defined as:

```
[language[_territory[.codeset]][@modifier]]
```

For example, if a user wanted to interact with the system in French, but required to sort German text files, *LANG* and *LC_COLLATE* could be defined as:

```
LANG=Fr_FR
LC_COLLATE=De_DE
```

This could be extended to select dictionary collation (say) by use of the "@modifier" field, e.g.,

```
LC_COLLATE=De_DE@dict
```

At run-time, these values are bound to a program's locale by calling the *setlocale()* function.

LOGNAME The name of the user's login account, corresponding to the login name from the user database (see <pwd.h>). For a value of *LOGNAME* to be portable across implementations of IEEE Std. 1003.1-1988, the value should be composed of characters from the portable filename character set.

NLSPATH Contains a sequence of templates which *catopen()* uses when attempting to locate message catalogues. Each template consists of an optional prefix, one or more substitution fields, a filename and an optional suffix.

For example:

```
NLSPATH="/system/nlslib/%N.cat"
```

defines that *catopen()* should look for all message catalogues in the directory */system/nlslib*, where the catalogue name should be constructed from the *name* parameter passed to *catopen* (*%N*), with the suffix *.cat*.

Substitution fields consist of a % symbol, followed by a single-letter keyword. The following keywords are currently defined:

<i>%N</i>	The value of the <i>name</i> parameter passed to <i>catopen()</i> .
<i>%L</i>	The value of <i>LANG</i> .
<i>%l</i>	The <i>language</i> element from <i>LANG</i> .
<i>%t</i>	The <i>territory</i> element from <i>LANG</i> .
<i>%c</i>	The <i>codeset</i> element from <i>LANG</i> .
<i>%%</i>	A single % character.

An empty string is substituted if the specified value is not currently defined. The separators "_" and "." are not included in *%t* and *%c* substitutions.

Templates defined in *NLSPATH* are separated by colons, ":". A leading or two adjacent colons "::" is equivalent to specifying *%N*. For example:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates to *catopen()* that it should look for the requested message catalogue in *name*, *name.cat* and */nlslib/\$LANG/name.cat*.

PATH The sequence of path prefixes that certain functions apply in searching for an executable file known only by a filename (a pathname that does not begin with a slash). The prefixes are separated by a colon, ":". When a non-zero-length prefix is applied to the filename, a slash is inserted between the prefix and filename. A zero-length prefix is a special prefix which indicates the current working directory. It appears as two

adjacent colons ":" as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. The list is searched from left to right until an executable program of the specified name is found. If the pathname being sought contains a slash, the search through the path prefixes is not performed.

TERM	The terminal type for which output is to be prepared. This information is used by commands and applications programs wishing to exploit special capabilities of the terminal.
TMPDIR	The pathname of a directory made available for programs that need a place to create a temporary file.
TZ	Timezone information. The contents of the environment variable named TZ are used by the functions <i>ctime()</i> , <i>localtime()</i> , <i>strftime()</i> and <i>mktime()</i> to override the default timezone. The value of TZ has the form:

`std offset [dst [offset], start [/time], end [/time]]`

Where:

std and *dst*

Three or more bytes that designate for the standard (*std*) and summer (*dst*) timezones. Only *std* is required, if *dst* is missing, then summer time does not apply in this locale. Upper- and lower-case letters are explicitly allowed. Any characters, except a leading colon ":", digits, a comma ",", a minus "-", a plus "+" or ASCII NUL, are allowed.

offset

Indicates the value one must add to the local time to arrive at Coordinated Universal Time. The offset has the form:

`hh [:mm [:ss]]`

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) is required and may be a single digit. The *offset* following *std* is required. If no *offset* follows *dst*, summer time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour must be between 0 and 24, and the minutes (and seconds) if present between 0 and 59. Out of range values may cause unpredictable behaviour. If preceded by a "-", the timezone is east of the Prime Meridian; otherwise it is west (which may be indicated by an optional preceding "+" sign).

start/time,end/time

Indicates when to change to and back from summer time, where *start/time* describes when the change from standard time to summer time occurs, and *end/time* describes when the change back happens. Each *time* field describes when, in current local time, the change is made.

The formats of *start* and *end* are one of the following:

Jn The Julian day n ($1 \leq n \leq 365$). Leap days are not counted. That is, in all years, February 28 is day 59 and March 1 is day 60. It is impossible to refer to the occasional February 29.

n The zero-based Julian day ($0 \leq n \leq 365$). Leap days are counted, and it is possible to refer to February 29.

Mm.n.d The d th day, ($0 \leq d \leq 6$) of week n of month m of the year ($1 \leq n \leq 5$, $1 \leq m \leq 12$), where week 5 means "the last d -day in month m " which may occur in either the fourth or the fifth week). Week 1 is the week in which the first day of the month falls. Day zero is Sunday.

Implementation specific defaults are used for *start* and *end* if these optional fields are not given.

The *time* has the same format as *offset* except that no leading sign ("-" or "+") is allowed. The default, if *time* is not given is 02:00:00.

Environment variable *names* used or created by an application should consist solely of characters from the portable filename character set. Other characters may be permitted by an implementation; applications must tolerate the presence of such names. Upper- and lower-case letters retain their unique identities and are not folded together. System defined environment variable names should begin with an upper-case letter or underscore, and be composed of only upper-case letters, digits and underscores.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name=value* pairs may be placed in the environment by manipulating the *environ* variable, or by using *envp* arguments when creating a process, see *exec* (in Volume 2, XSI System Interfaces and Headers).

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

<i>CDPATH</i>	<i>LC_CTYPE</i>	<i>MAILER</i>	<i>SHELL</i>
<i>COLUMNS</i>	<i>LC_MONETARY</i>	<i>MAILPATH</i>	<i>TERM</i>
<i>DEAD</i>	<i>LC_NUMERIC</i>	<i>MAILRC</i>	<i>TERMCAP</i>
<i>EDITOR</i>	<i>LC_TIME</i>	<i>MBOX</i>	<i>TERMINFO</i>
<i>EXINIT</i>	<i>LINES</i>	<i>NLSPATH</i>	<i>TMPDIR</i>
<i>HISTORY</i>	<i>LISTER</i>	<i>PAGER</i>	<i>TZ</i>
<i>HOME</i>	<i>LOGNAME</i>	<i>PATH</i>	<i>USER</i>
<i>IFS</i>	<i>LPDEST</i>	<i>PRINTER</i>	<i>VISUAL</i>
<i>LANG</i>	<i>MAIL</i>	<i>PS1</i>	
<i>LC_COLLATE</i>	<i>MAILCHECK</i>	<i>PS2</i>	

2.5 DEVICES

The names of specific I/O devices are known as “special files”. The mandatory devices, present in every XSI compliant system, are described in subsequent subsections. Most systems will contain other entries for specific devices.

2.5.1 */dev/null*

Data written on */dev/null* is discarded.

Reads from */dev/null* always return 0 bytes.

2.5.2 */dev/tty*

The file */dev/tty* is, in each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

2.6 OUTPUT DEVICES

Many of the commands and utilities are intended to be used directly by a user, rather than having their output processed by other programs.

Such utilities may make assumptions regarding the abilities of the terminal in use. For example, *pr* expects the terminal to have a bell if the *-p* option is invoked.

In some cases it is assumed that an asynchronous terminal is in use. On some systems, block-mode, synchronous terminals will cause those utilities anticipating asynchronous terminals to function in a modified manner, or possibly not at all. Because of the wide variety of terminals in use and the fact that the X/Open definition does not constrain the way in which any command is implemented, it is not possible to identify the commands affected in this way.

Application writers should not expect all terminals to support all possible attributes. An application wishing to use a wide set of attributes may restrict the number of terminals it can make use of.

2.7 SHELL BUILT-INS

Some commands are built into the shell, *sh*. Such commands have been marked in an **APPLICATION USAGE** on the relevant page.

These built-ins may not exist as separate executable commands accessible through the *system()*, *popen()* or one of the *exec* interfaces (as described in **Volume 2, XSI System Interfaces and Headers**). Some, for example, *cd*, cannot be implemented as anything other than a built-in.

The following are the relevant commands:

Shell built-in commands	
<i>cd</i>	<i>test</i>
<i>echo</i>	<i>umask</i>
<i>newgrp</i> †	<i>wait</i>
<i>pwd</i>	

† It should also be noted that, although not actually a built-in, on some implementations the command *newgrp* behaves like a built-in. Reference should be made to the entry for this command for further details.

Commands and Utilities

This chapter contains the definitions of the XSI Commands and Utilities. These are the standard utilities which are present on every X/Open compatible system and software development utilities which are present only on systems supporting the software development facility. The development utilities are marked (DEVELOPMENT) in their NAME lines.

NAME

`admin` – create and administer SCCS files (**DEVELOPMENT**)

SYNOPSIS

```
admin [-n] [-i [name]] [-rrel] [-t [name]] [-fflag [flag-val]] [-dflag [flag-val]]
      [-alogin] [-eloglein] [-m [mrlist]] [-y [comment]] [-h] [-z] file ...
```

DESCRIPTION

The *admin* utility is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of options and named files (note that SCCS filenames must begin with s.). If a named file does not exist, it is created, and its parameters are initialised according to the specified options. Parameters not initialised by an option are assigned a default value. If a named file does exist, parameters corresponding to specified options are changed, and other parameters are left as is.

If *file* is a directory, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The options are as follows. Each is explained as though only one named file is to be processed since the effects of the options apply independently to each named file.

-n This option indicates that a new SCCS file is to be created.

-i[*name*]

The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** option for delta numbering scheme). If the **-i** option is used, but the filename is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this option is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **-i** option is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **-i** option). Note that the **-i** option implies the **-n** option.

-rrel The *release* into which the initial delta is inserted. This option may be used only if the **-i** option is also used. If the **-r** option is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[*name*]

The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **-t** option is used and *admin* is creating a new SCCS file (the **-n** and/or **-i** options also used), the descriptive text filename must also be supplied. In the case of existing SCCS files:

- A **-t** option without a filename causes removal of descriptive text (if any) currently in the SCCS file.
- A **-t** option with a filename causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-f*flag* This option specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **-f** options may be supplied on a single *admin* command line. The allowable flags and their values are:

- b** Allow use of the **-b** option on a *get* command to create branch deltas.
- cceil** The highest release (i.e., ceiling), a number less than or equal to 9999, which may be retrieved by a *get* command for editing. The default value for an unspecified *c* flag is 9999.
- ffloor** The lowest release (i.e., floor), a number greater than 0 but less than 9999, which may be retrieved by a *get* command for editing. The default value for an unspecified *f* flag is 1.
- dSID** The default delta number (SID) to be used by a *get* command.
- i[*str*]** Causes the message issued by *get* or *delta* to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string; however, the string must contain a keyword, and no embedded newlines.
- j** Allow concurrent *get* commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
- l*list*** A *list* of releases to which deltas can no longer be made (i.e., *get -e* against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= *<range>* | *<list>*, *<range>*
<range> ::= SID | a

The character *a* in the *list* is equivalent to specifying all releases for the named SCCS file.

n Cause *delta* to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

qtext User definable *text* substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*.

mmod Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*. If the *m* flag is not specified, the value assigned is the name of the SCCS file with the leading "s." removed.

ttype The *type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*.

v[pgm] Cause *delta* to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the *m* option must also be used even if its value is null.)

-dflag Cause removal (deletion) of the specified *flag* from an SCCS file. The *-d* option may be specified only when processing existing SCCS files. Several *-d* options may be supplied on a single *admin* command. See the *-f* option for allowable *flag* names. (The *l list* flag gives a *list* of releases to be "unlocked". See the *-f* option for further description of the *l* flag and the syntax of a *list*.)

-a login A *login* name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several *a* options may be used on a single *admin* command line. As many *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a *!*, the users so specified are denied permission to make deltas.

-e login A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several *-e* options may be used on a single *admin* command line.

-y[comment]

The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*. Omission of the *-y* option results in a default comment line being inserted in the form:

date and time created YY/MM/DD HH:MM:SS by login

The *-y* option is valid only if the *-i* and/or *-n* options are specified (i.e., a new SCCS file is being created).

-m [mrlist]

The list of modification request (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*. The *v* flag must be set and the MR numbers are validated if the *v* flag has a value (the name of an MR number validation program). Diagnostics will occur if the *v* flag is not set or MR validation fails.

-h

Cause *admin* to check the structure of the SCCS file, and compare the newly computed checksum (the sum of all the characters in the SCCS file except those in the first line) with the checksum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This option inhibits writing on the file, so that it nullifies the effect of any other options supplied. It is only meaningful when processing existing files.

-z

The SCCS file checksum is recomputed and stored in the first line of the SCCS file (see *-h*, above).

Note that use of this option on a truly corrupted file may prevent future detection of the corruption.

All SCCS filenames must be of the form **s.filename**. New SCCS filenames are given read-only permission mode. Write permission in the parent directory is required to create a file. All writing done by *admin* is to a temporary *x-file*, named **x.filename** (see *get*) created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the *x-file* is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occur.

The *admin* utility also uses a transient lock file (named **z.filename**), which is used to prevent simultaneous updates to the SCCS file. See *get* for further information.

APPLICATION USAGE

It is recommended that directories containing SCCS files be writable by the owner only, and that SCCS files themselves be read-only. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

SEE ALSO

delta, get, prs, what.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

ar – archive and library maintainer

SYNOPSIS

ar [-drqtpmx] [posname] afile [name ...]

DESCRIPTION

The *ar* utility maintains groups of files combined into a single archive file. It is used to create and update library files as used by the link editor, see *ld*. It can be used, however, for any similar purpose. If an archive file is created from printable files, the entire archive file is printable.

When *ar* creates an archive file, it creates administrative information in a format that is portable across all machines. When there is at least one object file that *ar* recognises as such in the archive, an archive symbol table is created in the archive file and maintained by *ar*; it is used by the link editor to search the archive file. Whenever the *ar* utility is used to create or update the contents of such an archive, the symbol table is rebuilt. The *s* modifier character described below forces the symbol table to be rebuilt.

The argument *option* may be concatenated with one or more characters to modify the action. These modifier characters are taken from the set *vuabicl* but not all modifiers make sense with all options. See below for further explanation. The argument *posname* is the name of a file in the archive file, used for relative positioning; see options *-r* and *-m* below. The argument *afile* is the archive file. The *names* are constituent files in the archive file.

The meanings of the *option* characters are:

- d Delete the named files from the archive file. Valid modifiers are *vl*.
- r Replace the named files in the archive file. Valid modifiers are *vuabicl*. If the modifier *u* is used, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set *abi* is used, then the *posname* argument must be present, and specifies that new files are to be placed after (*a*) or before (*b* or *i*) *posname*. Otherwise new files are placed at the end.
- q Quickly append the named files to the end of the archive file. Valid modifiers are *vcl*. In this case *ar* does not check whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.
- t Print a table of contents of the archive file. If no names are given, all files in the archive are listed. If names are given, only those files are listed. Valid modifiers are *vs*. The *v* modifier gives a long listing of all information about the files.
- p Print the named files from the archive. Valid modifiers are *vs*.

- m Move the named files to the end of the archive. Valid modifiers are *vabil*. If a positioning modifier from the set *abi* is present, then the *posname* argument must be present and, as with the option *-r*, it specifies where the files are to be moved.
- x Extract the named files. If no names are given, all files in the archive are extracted. The archive file is not changed unless the modifier *s* is specified. Valid modifiers are *vs*.

The meanings of the modifier characters are:

- v Give verbose output. When used with the option *-d*, *-r*, *-q* or *-m* this gives a verbose file-by-file description of the making of a new archive file from the old archive (if one exists) and the constituent files. When used with *-x*, this precedes each file with its name.
- c Suppress the message that is produced by default when the archive file *afile* is created.
- l Place temporary files in the local current working directory, rather than in the directory specified by the environment variable *TMPDIR* or in the default directory.
- UN s Force the regeneration of the archive symbol table even if *ar* is not invoked with an option which will modify the archive file contents. This option is useful to restore the archive symbol table after it has been stripped, see *strip*.

This utility operates in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

Environment Variables

- IN *LC_TIME* determines the format of date and time strings output when listing the contents of an archive with the *v* modifier, e.g.,

```
ar -tv libc.a
```

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

SEE ALSO

cc, *strip*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit manner has been noted. The operation of this utility in an internationalised environment has been described.

NAME

at, batch – execute commands at a later time

SYNOPSIS

at time [date] [+ increment]

OF

at -r job ...

at -l [job ...]

batch

DESCRIPTION

The *at* and *batch* utilities read commands from standard input to be executed at a later time. The utility *at* allows the user to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits.

at

OF

The option *-r* to *at* removes jobs previously scheduled by *at* or *batch*. The job number is the number reported at invocation by *at* or *batch*. Job numbers can also be obtained by using the *-l* option. Only a process with the appropriate privileges is allowed to remove another user's jobs.

Standard output and standard error output are mailed to the user unless they are redirected. The environment variables, current directory, *umask* and *ulimit* are retained when the commands are executed. Open files, traps and priority are lost.

Users are permitted to use *at* if their name appears in the file */usr/lib/cron/at.allow*. If that file does not exist, the file */usr/lib/cron/at.deny* is checked to determine if the user should be denied access to *at*. If neither file exists, only a process with the appropriate privileges is allowed to submit a job. If only *at.deny* exists and is empty, global usage is permitted. The *at.allow* and *at.deny* files consist of one user name per line.

The *time* may be specified as 1, 2 or 4 digits. One- and two-digit numbers are taken to be hours, four digits to be hours and minutes. Alternately, the time may be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix *am* or *pm* may be appended; otherwise a 24-hour clock time is understood. The suffix *zulu* may be used to indicate GMT. The special names *noon*, *midnight*, *now* and *next* are also recognised.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by a comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", *today* and *tomorrow*, are recognised. If no *date* is given, *today* is assumed if the given hour is greater than the current hour and *tomorrow* is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is a number suffixed by one of the following: *minutes*, *hours*, *days*, *weeks*, *months* or *years*. (The singular form is also accepted.)

Thus legitimate commands include:

- at 0815am Jan 24
- at 8:15am Jan 24
- at now + 1 day
- at 5 pm Friday

OF The commands *at* and *batch* write the job number and schedule time to standard error.

batch

The utility *batch* submits a batch job. It is almost equivalent to *at now*, but not quite. For one, the job goes into a different queue. For another, *at now* does not work: it is too late (and results in an error message).

EXAMPLES

The *at* and *batch* utilities read from standard input the commands to be executed at a later time. It may be useful to redirect standard output within the specified commands.

1. This sequence can be used at a terminal:

```
batch
spell filename >outfile
<EOF>
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<!
spell filename 2>&1 >outfile | mail loginid
!
```

3. To have a job reschedule itself, *at* can be invoked from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

APPLICATION USAGE

Commands will be executed using *sh*.

SEE ALSO

crontab, *umask*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

References to "super-user" have been changed to "process with appropriate privileges", otherwise functionally equivalent to the entry in Issue 2.

NAME

awk – pattern-directed scanning and processing language

SYNOPSIS

```
awk [-Fc] program [parameters] [file ... ]  
awk [-Fc] -f progfile [parameters] [file ... ]
```

DESCRIPTION

The *awk* utility executes programs written in the *awk* programming language, which is specialised for data manipulation. An *awk* program is a sequence of patterns and corresponding actions. When input is read that matches a pattern, the action associated with that pattern is carried out.

The *file* arguments contain the input to be read. If no files are given or the filename – is given, the standard input is used.

PI The *awk* utility accepts 8-bit characters in data being processed, program string literals and program comments. Support of 8-bit characters in variable names is implementation defined.

Each line of input is matched in turn against the set of patterns in the program. The *awk* program may either be in a file *progfile* or may be specified in the command line, taking into account the quoting rules of the command interpreter.

Each line of input is matched in turn against each pattern in the program. For each pattern matched, the associated action is executed.

The *awk* utility interprets each input line as a sequence of fields where, by default, a field is a string of non-space, non-tab characters. This default whitespace field delimiter can be changed by using the *-Fc* option, or the variable FS, see below. The command *awk* denotes the first field in a line *\$1*, the second *\$2*, and so forth. *\$0* refers to the entire line. Setting any other field causes the re-evaluation of *\$0*.

Pattern-action statements in an *awk* program have the form:

```
pattern { action }
```

In any pattern-action statement, either the pattern or the action may be omitted. A missing action means print the input line to the standard output; a missing pattern is always matched, and its associated action is executed for every input line read.

Patterns

Patterns are *special patterns* or arbitrary Boolean combinations (!, ||, && and parentheses) of regular expressions and relational expressions. The operator ! has the highest precedence, then && and then ||. Evaluation is left to right and stops when truth or falsehood has been determined.

Boolean Operator	Meaning
!	negation
&&	and
	or

Special Patterns

The *awk* utility recognises two special patterns, **BEGIN** and **END**. **BEGIN** is matched once and its associated action executed before the first line of input is read. **END** is matched once and its associated action executed after the last line of input has been read. (See Examples 5 and 8.) These two patterns must have associated actions.

Relational Expressions

A pattern may be any expression that compares strings of characters or numbers. A relational expression is either

expression relational-operator expression

or

expression matching-operator regular-expression

The six relational operators are shown in the table below; regular-expression matching operators are described later. In a comparison, if both operands are numeric, a numeric comparison is made, otherwise a string comparison is made.

Relational Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
==	equal to

Regular Expressions

IN

The *awk* utility makes use of extended regular expression syntax. On systems supporting an internationalised environment, extended internationalised regular expression syntax is used (see **Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions**).

A regular expression must be surrounded by slashes. If *re* is a regular expression, then the pattern

/re/

matches any line of input that contains a substring specified by the regular expression. A regular expression comparison may be limited to a specific field by one of the two regular expression-matching operators, `~` and `!~`.

`$4 ~ /re/`

matches any line in which the fourth field matches the regular expression `/re/`.

`$4 !~ /re/`

matches any line in which the fourth field does not match the regular expression `/re/`.

Pattern Ranges

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the following occurrence of the second pattern.

Variables and Special Variables

Variables may be used in an *awk* program by assigning to them. They do not need to be declared. Like field variables, all variables are treated as string variables unless used in a clearly numeric context (see **Relational Expressions**). Field variables are designated by a `$` followed by a number or numerical expression. New field variables may be created by assigning a value to them. If not initialised, variables default to the empty string or zero. Other special variables set by *awk* are shown in the table below.

Special Variable	Meaning
<code>\$n</code>	The string read as field <code>n</code> .
<code>FS</code>	Input field separator. Set to tab and space by default.
<code>RS</code>	Input record separator. Set to newline by default.
<code>FILENAME</code>	Name of the current input file.
<code>NF</code>	Number of fields in the current record.
<code>NR</code>	Ordinal number of the current record from the start of input.
<code>OFMT</code>	<i>Print</i> statement output format for numbers. <code>%.6g</code> by default.
<code>OFS</code>	<i>Print</i> statement output field separation. One space by default.
<code>ORS</code>	<i>Print</i> statement output record separator. Newline by default.

Variable assignments of the form `x=value` may occur on the command line as parameters.

Actions

An action is a sequence of statements. A statement can be one of the following. Square brackets indicate optional elements.

```

if ( expression ) statement [ else statement ]
while ( expression ) statement
for ( expression ; expression ; expression ) statement
for ( variable in array ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
print [ expression-list ] [ >>expression ]
print [ expression-list ] [ |expression ]
printf format [ , expression-list ] [ >expression ]
printf format [ , expression-list ] [ >>expression ]
printf format [ , expression-list ] [ |expression ]
next
exit ( expression )

```

Any single statement may be replaced by a statement list enclosed in braces. The statements in a statement list are separated by newlines or semicolons. The symbol # anywhere in a program line begins a comment, which is terminated by the end of the line.

Statements are terminated by semicolons, newlines or right braces. A long statement may be split across several lines by ending each partial line with a \. An empty expression list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators + (addition), - (subtraction), * (multiplication), / (division), % (modulus operator) and concatenation (indicated by a blank between strings in an expression). The C-language operators ++, --, +=, -=, *=, /= and %= are also available in expressions. Variables may be scalars, array elements (denoted *x[i]*) or fields. Variables are initialised to the empty string. Array subscripts may be any string, not necessarily numeric.

String constants are surrounded by double quotes ("..."). A string expression is created by concatenating constants, variables, field names, array elements, functions and other expressions.

The *expression* acting as the conditional in an *if* statement can include the relational operators, the regular expression-matching operators, logical operators, juxtaposition for concatenation and parentheses for grouping. *Expression* is evaluated and if it is non-zero and non-null, *statement* is executed; otherwise, if *else* is present, the *statement* following the *else* is executed.

The *while*, *for*, *break* and *continue* statements are as in C-language.

The *print* statement prints its arguments on the standard output separated by the current output field separator (see variable OFS above) and terminated by the

output record separator (see variable `ORS` above). The `printf` statement formats its expression list according to *format*, see `printf()` in **Volume 3, XSI System Interfaces and Headers**. Output can be redirected to a file if `>expression` is present, appended (`>>expression`) or sent to a pipe (`|expression`).

The `next` statement causes the next input line to be scanned, skipping the remaining characters on the current input line. The `exit` statement causes the termination of the `awk` program, skipping the rest of the input.

Built-In Functions

The built-in function `length(s)` returns the length of its arguments taken as a string, or of the whole line, `$0`, if there is no argument.

There are also built-in functions `exp(x)` (the exponential function of x), `log(x)` (natural logarithm of x), `sqrt(x)` (square root of x) and `int(x)` (truncates its argument to an integer).

The built-in function `split(s1, arr, s2)` assigns the fields of the string `s1` to successive elements of the array `arr`, using the characters in string `s2` as field separators. Assignments start at array element `arr[1]`. The built-in function `substr(s, p, n)` returns the substring of `s` of at most n bytes that begin at position p . The built-in function `index(s1, s2)` returns the leftmost position if the string `s2` occurs in `s1` or zero if `s2` does not occur in `s1`.

The `getline` function immediately reads the next input record. Fields `NR` and `$0` are all set, but control is left at exactly the same spot in the `awk` program. The `getline` function returns 0 on end-of-file, and 1 otherwise.

The function `sprintf(fmt, expr, expr, ...)` formats the *expressions* according to the `printf()` format given by `fmt` and returns the resulting string.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed regular expressions, `LC_COLLATE` determines the behaviour of ranges, equivalence classes and multi-character collating elements, and `LC_CTYPE` determines the behaviour of character classes. `LC_COLLATE` determines the behaviour of the relational operators when performing comparisons on string values.

If either `LC_COLLATE` or `LC_CTYPE` are not set in the environment or are set to the empty string, the value of `LANG` will be used as a default for each unset/empty variable. If `LANG` is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXAMPLES

The following are examples of simple *awk* programs:

1. Print on the standard output all input lines for which field 3 is greater than 5.

```
$3 > 5
```

2. Print every tenth line.

```
(NR % 10) == 0
```

3. Print any line with a substring matching the regular expression.

```
/(G|D)(2[0-9][a-zA-Z]*)/
```

4. Print any line with a substring containing a G or D, followed by a sequence of digits and characters. This example uses character classes *digit* and *alpha* to match language-independent digit and alphabetic characters respectively.

```
/(G|D)([[:digit:][:alpha:]]*)/
```

5. Print the second to the last and the last field in each line. Separate the fields by a colon.

```
{OFS=":"; print $(NF-1), $NF}
```

6. Print the line number and number of fields in each line. The three strings representing the line number, the colon and the number of fields are concatenated and that string is printed.

```
BEGIN {line = 0}
{line = line + 1
 print line ":" NF}
```

7. Print lines longer than 72 characters.

```
length > 72
```

8. Print the first two fields in opposite order separated by the OFS.

```
{ print $2, $1 }
```

9. Add up the first column, print sum and average.

```
{s += $1 }
END {print "sum is ", s, " average is", s/NR}
```

10. Print fields in reverse order.

```
{ for (i = NF; i > 0; --i) print $i }
```

11. Print all lines between occurrences of the strings *start* and *stop*.

```
/start/, /stop/
```

12. Print all the lines in which the first field differs that of the previous line.

```
$1 != prev { print; prev = $1 }
```

13. Print file, filling in page numbers starting at 5.

```
/Page/ { $2 = n++; }  
      { print }
```

14. Command line.

```
awk -f program n=5 input
```

APPLICATION USAGE

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, 0 should be added to it; to force it to be treated as a string concatenate the empty string ("") to it.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

banner – make large letters

SYNOPSIS

OF **banner string ...**

DESCRIPTION

The *banner* utility writes each argument *string* in large letters (across the page) to the standard output, putting each argument on a separate "line". Spaces can be included in an argument by surrounding it with quotes. The maximum number of bytes that can be accommodated in a line is implementation dependent; however, a minimum of 10 can be assumed; excess characters are simply ignored.

APPLICATION USAGE

The font of the output is that of the US-ASCII character set.

SEE ALSO

echo.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

basename, dirname – deliver portions of pathnames

SYNOPSIS

basename string [suffix]

dirname string

DESCRIPTION

basename

The *basename* utility deletes any prefix ending in / from *string*. If the *suffix* is specified and present in *string*, the *suffix* is deleted from the *string*. The *basename* utility writes the result to the standard output.

dirname

The *dirname* utility delivers all but the last level of the pathname given in *string*.

EXAMPLES

1. The following example moves the file named **abc** to a file named **xyz** in the current directory:

```
mv abc `basename /p/q/xyz.c .c`
```

2. The following example will set the variable NAME to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/xyz.c`
```

APPLICATION USAGE

The *basename* and *dirname* utilities are normally used inside substitution marks (``) within command procedures.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cal – print calendar

SYNOPSIS

LA OF cal [[month] year]

DESCRIPTION

The *cal* utility writes a calendar for the specified *year* on the standard output. If *month* is also specified, a calendar for that month only is printed. If neither is specified, a calendar for the present month is printed. The argument *year* can be between 1 and 9999. (Note that *cal 83* refers to A.D. 83, not 1983.) The *month* is a number between 1 and 12.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

calendar – reminder service

SYNOPSIS

calendar

DESCRIPTION

The *calendar* utility consults the file **calendar** in the current directory and writes lines that contain today's or tomorrow's date anywhere in the line on the standard output. Month-day date formats such as *Aug. 24*, *august 24* and *8/24* are recognised. On weekends, *tomorrow* extends through Monday.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cat – concatenate and print files

SYNOPSIS

cat [-s] [file ...]

DESCRIPTION

The *cat* utility reads each *file* in sequence and writes it to the standard output. Thus:

cat file

writes the file on the standard output and:

cat file1 file2 >file3

concatenates the first two files and writes the result to the third.

If no input file is given, or if the argument - is encountered, *cat* reads from the standard input.

PI The -s option makes *cat* silent about non-existent files.

This utility operates in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

APPLICATION USAGE

Because of the mechanism used to perform output redirection, a command such as this:

cat file1 file2 >file1

will cause the original data in *file1* to be lost.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

cc – C-language compiler

SYNOPSIS

cc [options] file ...

DESCRIPTION

The *cc* utility is the interface to the C-language compilation system. The system conceptually consists of a preprocessor, compiler, optimiser, assembler and link editor. The *cc* utility processes the supplied *options* and then executes the various tools with the appropriate arguments.

The suffix of *file* indicates how it is to be treated. Files whose names end with *.c* are taken to be C-language source programs, and may be preprocessed, compiled, optimised and link edited. The compilation process may be stopped after the completion of any pass if the appropriate *options* are supplied. If the compilation process is allowed to complete the assembly phase, then an object program is produced; the object program for a source file called *xyz.c* is created in a file called *xyz.o*. However, the *.o* file is normally deleted if a single C-language program is compiled and link edited all at one go.

In the same way, arguments whose names end with *.s* are taken to be assembly source programs, and may be assembled and link edited. Files with names ending in *.i* are taken to be preprocessed C-language source programs and may be compiled, optimised, assembled and link edited. Files whose names do not end in *.c*, *.s* or *.i* are handed to the link editor.

By default, if an executable file is produced (i.e., the link-edit phase is allowed to complete) the file is called *a.out*. This default name can be changed with the *-o* option (see below).

The following options are interpreted by *cc*:

- c* Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.
- f* Include floating point support for systems without an automatically included floating point implementation. This option is ignored on systems that do not need it.
- PI OP *-g* Cause the compiler to generate additional information needed for use by a debugger (possibly *sdb*).
- o outfile*
Use the name *outfile* instead of the default *a.out* for the executable file produced. This is a link-edit option.
- PI OP *-p* This option is reserved for invoking implementation-specific profiling procedures.
- PI *-q* This option is reserved for specifying implementation-specific profiling directives.

- UN **-E** Run only the preprocessor on the named C-language programs and send the result to the standard output.
- PI **-F** This option is reserved for implementation-specific optimisation directives.
- O** Do compilation phase optimisation. This option will not affect *.s* files.
- UN **-P** Run only the preprocessor on the named C-language programs and leave the result on corresponding files suffixed *.i*.
- UN **-S** Compile and do not assemble the named C-language programs, and leave the assembler-language output on corresponding files suffixed *.s*.
- PI **-Wc, arg[, arg ...]**
Pass the argument[s] *arg* to phase *c* where *c* is one of [*p02al*] indicating preprocessing, compiling, optimising, assembling or link editing, phases respectively. For example, *-Wa, -m* passes *-m* to the assembler phase.

The *cc* utility also recognises a number of options which it will pass (with their associated arguments) directly to another phase of the *cc* utility. The use of the *-W* option is not required for these options.

The following options are passed by *cc* (with their associated arguments) to the preprocessor phase:

- C** By default, the preprocessor strips C-language style comments. If the *-C* option is specified, all comments (except those found on preprocessor directive lines) are passed along.
- U name**
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor.
- D name**
- D name=def**
Define *name* as if by a **#define** directive. If no *=def* is given, *name* is defined as 1. The *-D* option has lower precedence than the *-U* option.
- I dir** Change the algorithm for searching for **#include** files whose names do not begin with */* to look in *dir* before looking in the directories in the standard list. Thus, **#include** files whose names are enclosed in *" "* will be searched for first in the directory of the file with the **#include** line, then in the directories named in the *-I* options, and then last in the standard list. For **#include** files whose names are enclosed in *< >*, the directory for the file with the **#include** line is not searched.

The following options are passed by *cc* (with their associated arguments) to the link-edit phase:

- e epsym**
Set the default entry point address for the output file to be that of the symbol

epsym.

- l *xxx*
Search the library which has the abbreviation *xxx*. A library is searched when its name is encountered, so the placement of the -l option is significant.
- r
Retain relocation entries in output object file. Relocation entries must be saved if the output is to become the input of a subsequent *cc* run. The link-edit phase will not complain about unresolved references and will not make the object output executable.
- s
Strip all symbolic references from the output object file.
- u *symname*
Enter *symname* as an undefined symbol into the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force loading of the first routine.
- L *dir*
Change the algorithm for searching for the library *xxx* to look in *dir* before looking in the default library directories. This option is only effective if it precedes the -l option on the command line.

Other arguments are taken to be C-language compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-language compatible routines, and are passed directly to the link editor. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name **a.out** (unless the -o link-edit option is used).

The standard C-language library is automatically available to the C-language program. Other libraries may be specified explicitly using the -l option with *cc*.

Users may specify, by means of the *TMPDIR* environment variable, the directory in which any temporary files are to be created.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Since the *cc* utility usually creates files in the current directory during the compilation process, it is typically necessary to run the *cc* utility in a directory in which a file can be created.

SEE ALSO

sdb.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

The *cd* command changes the current working directory, that is, the starting point for path searches for pathnames not beginning with /. If *directory* is not specified, the value of the environment variable *HOME* is used as the new working directory. If *directory* specifies a complete path starting with /, . or .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the *CDPATH* environment variable. *CDPATH* has the same syntax as, and similar semantics to, the *PATH* variable, see *sh*. The *cd* command must have execute (search) permission in *directory*.

APPLICATION USAGE

This is always a shell built-in command.

SEE ALSO

pwd, *sh*, XSI System Interfaces and Headers, *chdir()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cf_{low} – generate C-language flowgraph (DEVELOPMENT)

SYNOPSIS

cf_{low} [-r] [-ix] [-i_] [-dnum] file ...

DESCRIPTION

The *cf_{low}* utility analyses a collection of C-language, *yacc*, *lex*, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in *.y*, *.l*, *.c* and *.i* are *yacc*'d, *lex*'d and preprocessed by the *cc* preprocessor phase (bypassed for *.i* files) as appropriate, and then run through the first pass of *lint*. (The *-I*, *-D* and *-U* options of the *cc* preprocessor phase are also understood by *cf_{low}*.) Files suffixed with *.s* are assembled and information is extracted (as in *.o* files) from the symbol table. The output of all this processing is collected and turned into a graph of external references, which is then displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable amount of indentation indicating the level. This is followed by the name of the global, a colon and its definition. Normally globals are only functions not defined as an external or beginning with an underscore; see below for the *-i* inclusion option. For information extracted from C-language source, the definition consists of an abstract type declaration (e.g., ***) and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the filename and location counter under which the symbol appeared (e.g., *text*).

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only *<>* is printed.

The following options are interpreted by *cf_{low}*:

- r Reverse the caller:callee relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.
- ix Include external and static data symbols. The default is to include only functions in the flowgraph.
- i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).
- dnum

The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number (typically greater than 32000). Attempts to set the cutoff depth to a non-positive integer will be ignored.

EXAMPLE

Given the following in **file.c**:

```
int i;

main()
{
    f();
    g();
    f();
}

f()
{
    i = h();
}
```

The command

```
cflow -ix file.c
```

produces the output

```
1      main: int(), <file.c 4>
2      f: int(), <file.c 11>
3          h: <>
4          i: int, <file.c 1>
5      g: <>
```

APPLICATION USAGE

Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

SEE ALSO

cc, *lex*, *lint*, *yacc*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

chmod – change mode

SYNOPSIS

chmod *mode* *file* ...

DESCRIPTION

The permissions of *file* or files are changed according to *mode*, which may be absolute or symbolic.

An absolute *mode* is a four-octal-digit number constructed from the logical-or (sum) of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	reserved
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0040	read by group
0020	write by group
0010	execute (search) by group
0004	read by others
0002	write by others
0001	execute (search) by others

A symbolic *mode* has the form:

[*who*] *op* *permission* [*op* *permission*]

The *who* part is a combination of the letters *u* (user), *g* (group) and *o* (other). The letter *a* stands for *ugo*, the default if *who* is omitted.

The argument *op* can be +, to add *permission* to the file's mode, -, to take away *permission*, or =, to assign *permission* absolutely (all other bits will be reset).

The argument *permission* is any combination of the letters *r* (read), *w* (write), *x* (execute) and *s* (set user or group ID); *u*, *g* or *o* indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter *s* is only useful with *u* or *g*.

Only the owner of a file (or a process with appropriate privileges) may change its mode. In order to set set-group-ID, the group of the file must correspond to the user's current group ID.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXAMPLES

1. This example denies write permission to others:

```
chmod o-w file
```

2. This example makes a file executable:

```
chmod +x file
```

SEE ALSO

ls, *umask*, **Volume 2, XSI System Interfaces and Headers**, *chmod()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

The *chown* utility changes the user ID of *file* to *owner*. The owner may be either a decimal user ID or a login name found in the User Database.

The *chgrp* utility changes the group ID of *file* to *group*. The group may be either a decimal group ID or a group name found in the Group Database.

Unless either command is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file are cleared upon successful completion of either utility.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some systems restrict the use of *chown* to a user with appropriate privileges.

Some systems restrict the use of *chgrp* to a user with appropriate privileges when the *group* specified is not the effective group ID or one of the supplementary group IDs of the calling process.

SEE ALSO

chmod, **Volume 2, XSI System Interfaces and Headers**, *chown()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

chroot – change root directory for a command

SYNOPSIS

chroot newroot command

DESCRIPTION

The *chroot* utility executes the given *command*, with its root directory set to *newroot*. The meaning of any initial slashes, / , in pathnames is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

This command is usually restricted to the processes with appropriate privileges.

Notice that:

chroot newroot command >x

will create the file x relative to the original root, not the new one.

The *newroot* argument is relative to the current root of the running process, even if a *chroot* is currently in effect.

APPLICATION USAGE

The user should note that many applications reference additional files, which will need to be present in the new file tree.

SEE ALSO

Volume 2, XSI System Interfaces and Headers, *chdir()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cmp – compare two files

SYNOPSIS

OF **cmp** [-l] [-s] file1 file2

DESCRIPTION

The *cmp* utility compares two files. If *file1* is -, the standard input is used.

OF Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is identical to the first part of the other, then it is reported that end-of-file was reached in the shorter file (before any differences were found).

The options are:

OF -l Print the byte number (decimal) and the differing bytes (octal) for each difference.

-s Print nothing for differing files; return codes only.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXIT STATUS

Exit status is:

- 0 identical files
- 1 different files
- 2 inaccessible file or missing argument

APPLICATION USAGE

Byte numbering begins at 1, rather than at 0.

SEE ALSO

comm, *diff*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

col – filter reverse line-feeds

SYNOPSIS

col [-bfp_x]

DESCRIPTION

The *col* utility reads from the standard input and writes to the standard output. It performs the line overlays implied by reverse line-feeds, and by forward and reverse half-line-feeds.

If the *-b* option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

Although *col* accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the *-f* (fine) option; in this case, the output from *col* may contain forward half-line-feeds, but will still never contain either kind of reverse-line motion.

Unless the *-x* option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO and SI are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are <space>, <backspace>, <tab>, <return>, <newline>, SI, SO, VT, reverse line-feed, forward half-line-feed and reverse half-line-feed. The VT character is an alternative form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

The ASCII codes for the control functions and line-motion sequences mentioned above are as given in the table below. ESC stands for the ASCII *escape* character, with the octal code 033; ESC-*x* means a sequence of two characters, ESC followed by the character *x*.

reverse line-feed	ESC-7
reverse half-line-feed	ESC-8
forward half-line-feed	ESC-9
vertical tab (VT)	013
start-of-text (SO)	016
end-of-text (SI)	017

Normally, *col* will remove any escape sequences found in its input that are unknown to it; the *-p* option may be used to force these to be passed through unchanged. The use of this option is discouraged unless the user is aware of the consequences.

APPLICATION USAGE

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

OF **comm [-123]] file1 file2**

DESCRIPTION

The *comm* utility reads *file1* and *file2*, which should be ordered in the collating sequence of *sort*, see *sort*, and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename – means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus *comm -12* prints only the lines common to the two files; *comm -23* prints only lines in the first file but not in the second; *comm -123* is a no-op.

APPLICATION USAGE

IN In an internationalised environment, the value of the *LC_COLLATE* environment variable must be equal to the value it had when the input files were sorted.

If *comm* does not support selection of collating sequences via *LC_COLLATE*, the input files must be sorted according to the collating sequence of the "C" locale (see Volume 3, XSI Internationalisation, Chapter 7, C Program Locale).

SEE ALSO

cmp, *diff*, *sort*, *uniq*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an internationalised environment has been described.

NAME

cp, ln, mv – copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target

ln [-f] file1 [file2 ...] target

mv [-f] file1 [file2 ...] target

DESCRIPTION

These utilities respectively copy, link or move files; *file1* and *target* may not be the same. If *target* is not a directory, then only one file may be specified before it; if *target* is an existing file, its contents are destroyed, otherwise (*target* is neither an existing file nor a directory) the file *target* is created. If *target* is a directory, then more than one file may be specified before it; the specified files are respectively copied, linked or moved to that directory.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

cp If *target* is not a directory, *cp* copies *file1* to *target*. If *target* exists, its contents are overwritten, but the mode, owner and group are not changed. If *target* is a link to a file, all links remain (the file is changed).

If *target* is a directory, then the specified files are copied to that directory. For each *file* a new file, with the same mode, is created in the target directory; the owner and the group are those of the user making the copy.

ln If *target* is not a directory, *ln* links *file1* to *target*; that is, the name *target* is linked to the file *file1*. If *target* exists and its mode forbids writing, the mode is printed and the user is asked for a response; if the response begins with a *y* or the locale's equivalent of a *y* (and the user is permitted), then the *ln* occurs. When the *-f* option is used or if the standard input is not a terminal, no questions are asked and the *ln* is done where permitted.

If *target* is a directory then the specified files are linked to that directory. That is, files with the same names are created in the directory linked to the specified files.

mv If *target* is not a directory, *mv* moves (renames) *file1* as directed. If *target* does not exist, and has the same parent as *file1*, *file1* may be a directory: this allows a directory rename.

If *target* is a directory, then the specified files are moved to that directory.

If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

IN If *target* is a file and its mode forbids writing, the mode is printed and the user is asked for a response; if the response begins with a *y*, or the locale's equivalent of a *y* (and the user is permitted), then the *mv* occurs. When the *-f* option is used or if the standard input is not a terminal, then no questions are asked and the *mv* is done where

permitted.

Environment Variables

IN *LANG* determines the locale's equivalent of *y* (for yes/no queries).

If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

If *file1* and *target* lie on different file systems, *mv* may achieve the move by copying the file and deleting the original. In this case any linking relationship with other files is lost.

The *ln* utility will not link across file systems.

The *mv* utility can only be used on directories for the purpose of renaming directories to a new name in the same parent. To rename directories to new parents, or to copy directories, use should be made of the *find* and *cpio* utilities.

SEE ALSO

chmod, *cpio*, *find*, *rm*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

cpio – copy file archives in and out

SYNOPSIS

```
cpio -o [ acBv ]  
cpio -i [ Bcdmrtuvf ] [ pattern ... ]  
cpio -p [ adlmruv ] directory
```

DESCRIPTION

cpio -o

The command *cpio -o* (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information. Output is padded to a 512-byte boundary.

cpio -i

The command *cpio -i* (copy in) extracts files from the standard input, which is assumed to be the product of a previous *cpio -o*. Only files with names that match *patterns* are selected. The arguments *patterns* are expressions making use of shell metanotation. In *patterns*, the metacharacters also match the / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous *cpio -o*. The owner and group of the files will be that of the current user unless the user has appropriate privileges, which causes *cpio* to retain the owner and group of the files of the previous *cpio -o*.

cpio -p

The command *cpio -p* (pass) reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

Options

The meanings of the available options are:

- a Reset access times of input files after they have been copied. (When option *l* (see below) is also specified, the access times of the linked files are not reset.)
- B Input/output is to be blocked 5120 bytes to the record (does not apply to the *-p* option; meaningful only with data directed to or from character special files).
- d Directories are to be created as needed.
- c Write or read header information in character form for portability.
- r Interactively rename files. If the user types a null line, the file is skipped.
- t Print a table of contents of the input. No files are created.
- u Copy unconditionally (normally, an older file will not replace a newer file with the same name).
- v Verbose: causes the names of the affected files to be printed. With the *t* option, provides a detailed listing.

- l Whenever possible, link files rather than copying them. Usable only with the `-p` option.
- m Retain previous file modification time. This option is ineffective on directories that are being copied.
- f Copy in all files except those in *patterns*.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed filename patterns (e.g., `'[a-i]*'`), `LC_COLLATE` determines the behaviour of ranges, equivalence classes and multi-character collating elements, and `LC_CTYPE` determines the behaviour of character classes. `LC_TIME` determines the format of date and time strings output when listing the contents of an archive with the `-v` option, e.g.,

```
cpio -icvt < /dev/sctmtm0
```

If `LC_COLLATE`, or `LC_CTYPE` or `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default for each unset/empty variable. If `LANG` is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXAMPLES

1. Copy the contents of a directory onto an archive:

```
ls | cpio -oc > ./cpio.out
```

2. Duplicate a directory hierarchy:

```
cd olddir
find . -depth -print | cpio -pd ../newdir
```

APPLICATION USAGE

Only a user with appropriate privileges can copy special files.

Archives created by *cpio* are portable between X/Open compatible systems provided the same procedures are used. See **Volume 3, XSI Source Code Transfer** for details.

Not all systems support the use of the shell metanotation `[! ...]` in *cpio* patterns, as a set negation character. Portable applications should avoid the use of this notation.

IN **ISO 8859-1:1987** is defined as the transmission codeset for communication between X/Open systems. However, 8-bit data and file names may not be portable to non-internationalised systems. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines, and that only characters defined in the Portable Filename Character Set are used for naming files.

FUTURE DIRECTIONS

The IEEE P1003.2 standard is currently proposing an alternative interface to data interchange media. It is possible that this utility will be withdrawn in a future issue if the functionality described here is subsumed into the new utility.

SEE ALSO

ar, find, ls, tar.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

crontab – user crontab file

SYNOPSIS

```
crontab [ file ]  
crontab -r  
crontab -l
```

DESCRIPTION

The *crontab* utility copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontab files. The *-r* option removes a user's crontab file from the crontab directory. The option *-l* will list the crontab file of the invoking user.

Users are permitted to use *crontab* if their names appear in the file */usr/lib/cron/cron.allow*. If that file does not exist, the file */usr/lib/cron/cron.deny* is checked to determine if the user should be denied access to *crontab*. If neither file exists, only a process with appropriate privileges is allowed to submit a job. If only *cron.deny* exists and is empty, global usage is permitted. The *cron.allow* and *cron.deny* files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

- minute [0, 59]
- hour [0, 23]
- day of the month [1, 31]
- month of the year [1, 12]
- day of the week ([0, 6] with 0 = Sunday)

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, each one is effective independent of the other. For example, *0 0 1,15 * 1* would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to *** (for example, *0 0 * * 1* would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the command interpreter at the specified times. A *%* character in this field (unless escaped by **) is translated to a newline character. Only the first line (up to a *%* or end of line) of the command field is executed by the command interpreter. The other lines are made available to the command as standard input. The event scheduler supplies a default environment, defining the environment variables *HOME*, *LOGNAME* and *PATH*.

Note: If standard output and standard error are not redirected, any generated output or errors will be mailed to the user.

APPLICATION USAGE

The new crontab file for a user overwrites an existing one.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

References to "super-user" have been changed to "process with appropriate privileges", otherwise functionally equivalent to the entry in Issue 2.

NAME

csplit – context split

SYNOPSIS

csplit [-s] [-k] [-f prefix] file arg1 [arg2 ... argn]

DESCRIPTION

The *csplit* utility reads *file* and separates it into $n+1$ sections, defined by the arguments *arg1*...*argn*. By default the sections are written to *xx00*... *xxn* (n may not be greater than 99). These sections will obtain the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.

·
·
·

- n : From the line referenced by *argn* to the end of *file*.

If the *file* argument is a -, then standard input is used.

The options to *csplit* are:

- s The *csplit* utility normally writes the character counts for each file created. If the -s option is present, *csplit* suppresses the printing of all character counts.
- k The *csplit* utility normally removes created files if an error occurs. If the -k option is present, *csplit* leaves previously created files intact.

-f*prefix*

If the -f option is used, the created files are named *prefix00* ... *prefixn*. The default is *xx00* ... *xxn*.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

/re/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *re*. Simple regular expression syntax is accepted. The current line becomes the line containing *re*. This argument may be followed by an optional + or - sign and a number indicating the number of lines (e.g., */Page/-5*).

%*re*%

This argument is the same as */re/*, except that no file is created for the section.

line_no

A file is to be created from the current line up to (but not including) the line number *line_no*. The current line becomes *line_no*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows an *re*-type argument, that argument is applied *num* more times. If it follows *line_no*, the file will be split every *line_no* lines (*num* times) from that point.

All *re*-type arguments that contain blanks or other characters meaningful to the command interpreter must be enclosed in the appropriate quotes. Regular expressions may not contain embedded newlines. The *csplit* utility does not affect the original file; it is the user's responsibility to remove it.

ERRORS

An error is reported if an argument does not reference a line between the current position and the end of the file.

EXAMPLES

1. This example creates four files, **cobol00** ... **cobol03**:

```
csplit -fcobol file '/PROCEDURE DIVISION/' /par5./ /par16./
```

After editing the split files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

2. This example splits the file at every 100 lines, up to 10000 lines:

```
csplit -k file 100 {99}
```

The *-k* option causes the created files to be retained if there are less than 10000 lines; however, an error message would still be printed.

3. Assuming that **prog.c** follows the normal C-language coding convention of ending routines with a **}** at the beginning of the line, this example will create a file containing each separate C-language function (up to 21) in **prog.c**.

```
csplit -k prog.c '%main(%' '/'^)/+1' {20}
```

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cu – call another system

SYNOPSIS

```
UN  cu [-s speed] [-l line] [-h] [-t] [-d] [-o | -e] [-n] telno
UN  cu [-s speed] [-h] [-d] [-o | -e] -l line
UN  cu [-h] [-d] [-o | -e] systemname
```

DESCRIPTION

The *cu* utility calls up another system. It manages an interactive conversation, with possible transfers of ASCII files.

The *cu* utility accepts the following options and arguments:

-s speed

Specify the transmission speed. The default value is "Any" speed which will depend on the order of the lines in the system devices file.

-l line

Specify a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the *-l* option is used without the *-s* option, the speed of a line is taken from the devices file. When the *-l* and *-s* options are both used together, *cu* will search the devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise, an error message will be printed and the call will not be made. If the specified device is associated with an autodialler, a telephone number must be provided. Use of this option with *systemname* rather than *telno* is not allowed (see *systemname* below).

-h Emulate local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t Used to dial an ASCII terminal which has been set to auto-answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-d Cause diagnostic traces to be printed.

-o Designate that odd parity is to be generated for data sent to the remote system.

-e Designate that even parity is to be generated for data sent to the remote system.

-n For added security, prompt the user to provide the telephone number to be dialled rather than taking it from the command line.

telno When using an automatic dialler, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname

A *uucp* system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from a system file.

Note: The *systemname* option should not be used in conjunction with the *-l* and *-s* options as *cu* will connect to the first available line for the system name specified ignoring the requested line and speed.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with *~*, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with *~*, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote system so that the buffer is not overrun. Lines beginning with *~* have special meanings.

The *transmit* process interprets the following user-initiated commands as:

- ~.* terminate the conversation.
- ~!* escape to an interactive command interpreter on the local system.
- ~!cmd...*
execute *cmd* on the local system.
- ~\$cmd...*
run *cmd* locally and send its output to the remote system for execution.
- ~%cd* change the directory on the local system.
- ~%take from [to]*
copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
- ~%put from [to]*
copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.
- ~~line* send the line *~line* to the remote system.
- ~%break*
transmit a BREAK to the remote system (which can also be specified as *~%b*).
- ~%nostop*
toggle between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output.

The use of *~%put* requires *stty* and *cat* on the remote side. It also requires that the current <erase> and <kill> characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of `~%take` requires the existence of *echo* and *cat* on the remote system. Also, tabs mode (see *stty*) should be set on the remote system if tabs are to be copied without expansion to spaces.

When *cu* is used on system *X* to connect to system *Y* and subsequently used on system *Y* to connect to system *Z*, commands on system *Y* can be executed by using `~^`. For example, *uname* can be executed on *Z*, *X*, *Y* as follows (the response is given in italics):

```
uname
Z
^[X]!uname
X
^^[Y]!uname
Y
```

In general, `~` causes the command to be executed on the original machine; `^^` causes the command to be executed on the next machine in the chain.

EXAMPLES

1. To dial a system whose telephone number is 9 1 201 555 1212 using 1200 baud (where dial tone is expected after the 9):

```
cu -s 1200 9=12015551212
```

If the speed is not specified, "Any" is the default value.

2. To login to a system connected by a direct line:

```
cu -l /dev/ttyXX
or
cu -l ttyXX
```

3. To dial a system with the specific line and a specific speed:

```
cu -s 1200 -l ttyXX
```

4. To dial a system using a specific line associated with an autodialler:

```
cu -l culXX 9=12015551212
```

5. To use a system name:

```
cu systemname
```

APPLICATION USAGE

Typical implementations of this utility require a communications line configured to use the *termios* (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) interface. On systems where none of these lines are available, this utility may not be present.

SEE ALSO

cat, echo, stty, uname, uucp.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut -clist [file ...]

cut -flist [-d char] [-s] [file ...]

DESCRIPTION

The *cut* utility cuts out columns from a table or fields from each line of a file. The fields as specified by *list* can be of fixed length, specified by character position (*-c* option), or the length can vary from line to line and be marked with a field delimiter character like tab (*-f* option). The *cut* utility can be used as a filter; if no *files* are given, the standard input is used.

The option qualifier *list* (see options *-c* and *-f* below) is a comma separated list of integers (in increasing order), with an optional “-” to indicate ranges; e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last).

The meanings of the options are:

-clist

The *list* following *-c* (no space) specifies character positions (e.g., -c1-72 would pass the first 72 characters of each line).

-flist

The *list* following *-f* is a list of fields assumed to be separated in the file by a delimiter character (see *-d*); e.g., -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings) unless *-s* is specified.

-dchar

The character following *-d* is the field delimiter (used with the *-f* option only). Default is the tab character. Space or other characters with special meaning to the command interpreter must be quoted.

-s Suppress lines with no delimiter characters when used with the *-f* option. Unless specified, lines with no delimiters will be passed through untouched.

Either the *-c* or the *-f* option must be specified.

APPLICATION USAGE

The *grep* utility can be used to make horizontal “cuts” (by context) through a file, and *paste* to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste* together.

SEE ALSO

grep, *paste*, *sh*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

cxref – generate C-language program cross-reference (DEVELOPMENT)

SYNOPSIS

cxref [options] file ...

DESCRIPTION

The *cxref* utility analyses a collection of C-language *files* and attempts to build a cross-reference table. Information from **#define** lines is included in the symbol table. A listing is produced on standard output of all symbols (auto, static and global) in each *file* separately, or with the *-c* option, in combination. Each symbol contains an asterisk before the declaring reference.

In addition to the *-D*, *-I* and *-U* options (which are identical to their interpretation by *cc*), the following options are interpreted by *cxref*:

- c* Print a combined cross-reference of all input files.
- wnum*
Width option that formats output no wider than *num* (decimal) columns. This option will default to 80 if *num* is not specified or is less than 51.
- o file*
Direct output to named *file*.
- s* Operate silently; does not print input filenames.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

date – print the date

SYNOPSIS

UN **date mmddhhmm[yy]**

date [+format]

DESCRIPTION

OF LA If no argument is given, the current date and time are printed.

UN If an argument in the form *mmddhhmm[yy]* is given, *date* attempts to set the system date from the value given in the argument. This is only possible if the user has appropriate privileges and the system permits the setting of the system date. The first *mm* is the month (number); *dd* is the day (number); *hh* is the hour (number, 24-hour system); the second *mm* is the minute (number) and *yy* is the last two digits of the year and is optional. For example:

```
date 10080045
```

sets the date to October 8, 00:45. The current year is the default if *yy* is omitted.

The system operates in GMT; *date* takes care of conversion to and from the local time. TZ is used to override the default timezone, see *ctime()*.

If an argument beginning with + is given, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to the *printf()* function, see **Volume 2, XSI System Interfaces and Headers, *printf()***. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a newline character.

Field descriptors:

IN	a	abbreviated weekday name
IN	A	full weekday name
IN	b	abbreviated month name
IN	B	full month name
LA	c	current date and time representation
	d	day of month – [01, 31]
	D	date as %m/%d/%y
IN	h	abbreviated month name
	H	hour – [00, 23]
LA	I	hour (12-hour clock) as a decimal number [01, 12]
	j	day of year – [001, 366]
	m	month of year – [01, 12]
	M	minute – [00, 59]
	n	insert a newline character
LA	p	equivalent of a.m. or p.m.
LA	r	time in a.m./p.m. notation
	S	second – [00, 61]

	t	insert a tab character
	T	time as %H:%M:%S
IN	U	week number of the year (Sunday as the first day of the week) as a decimal number [00, 53]
	w	day of week (Sunday = 0) – [0, 6]
IN	W	week number of the year (Monday as the first day of the week) as a decimal number [00, 53]
IN	x	current date representation
IN	X	current time representation
	y	last two digits of year – [00, 99]
IN	Y	year with century as a decimal number
IN	Z	timezone name, or no characters if no timezone exists

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN **LC_TIME** determines the content (e.g., weekday names produced by the field descriptor, *a*) and format (e.g., current time representation produced by the field descriptor, *x*) of date and time strings output by the *date* command. **TZ** determines the timezone name returned by the field descriptor, *Z*. If present, the environmental variable **TZ** is used to override the default timezone, see *ctime()*.

If **LC_TIME** is not set in the environment or is set to the empty string, the value of **LANG** will be used as a default. If **LANG** is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXAMPLE

The following generates output as shown:

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

Output:

```
DATE: 08/01/76
TIME: 14:45:05
```

APPLICATION USAGE

The range of values for *S* (seconds) is [0, 61] rather than [0, 59] to allow for the occasional leap second.

SEE ALSO

Volume 2, XSI System Interfaces and Headers, *ctime()*, *printf()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

dd – convert and copy a file

SYNOPSIS

dd [[option=value] ...]

DESCRIPTION

The *dd* utility copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

Option	Values
if=file	input filename; standard input is default.
of=file	output filename; standard output is default.
ibs=n	input block size <i>n</i> bytes (default 512).
obs=n	output block size (default 512).
bs=n	set both input and output block size, superseding <i>ibs</i> and <i>obs</i> ; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done.
cbs=n	conversion buffer size.
skip=n	skip <i>n</i> input blocks before starting copy.
seek=n	seek <i>n</i> blocks from beginning of output file before copying.
count=n	copy only <i>n</i> input blocks.
conv=ascii	convert EBCDIC to ASCII.
ebcdic	convert ASCII to EBCDIC.
ibm	slightly different map of ASCII to EBCDIC.
lcase	map alphabetics to lower-case.
ucase	map alphabetics to upper-case.
swab	swap every pair of bytes.
noerror	do not stop processing on an error.
sync	pad every input block to <i>ibs</i> .
...,...	several comma-separated conversions.

Where sizes are specified, a number of bytes is expected. A number may end with *k* or *b* to specify multiplication by 1024 or 512 respectively; a pair of numbers may be separated by *x* to indicate a product.

The option *cbs* is used only if ASCII or EBCDIC conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and <newline> added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

This example will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file x:

```
dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

APPLICATION USAGE

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC.

More than one variant of the EBCDIC character set exists.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

delta – make a delta (change) to an SCCS file (**DEVELOPMENT**)

SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] file ...

DESCRIPTION

The *delta* utility is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get* (called the *g-file*, or generated file).

The *delta* utility makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with *s.*) and unreadable files are silently ignored. If a name of *-* is given, the standard input is read; in this case the *-y* option (see below) is required on the command line; if the *-m* option (see below) would normally be required, then it too is required on the command line. Each line of the standard input is taken to be the name of an SCCS file to be processed.

The *delta* utility may issue prompts on the standard output depending upon certain options specified and flags, see *admin*, that may be present in the SCCS file (see *-m* and *-y* options below).

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS and will cause an error.

Keyletter arguments apply independently to each named file.

-rSID Uniquely identify which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding *gets* for editing (*get -e*) on the same SCCS file were done by the same person (login name). The SID value specified with the *-r* option can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* utility, see *get*. A diagnostic results if the specified SID is ambiguous, or if it is necessary and omitted on the command line.

-s Suppress the issue on the standard output of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n Specify retention of the edited *g-file* (normally removed at completion of delta processing).

-glist

Specify a *list*, see *get* for the definition of *list*, of deltas which are to be ignored when the file is accessed at the change level (SID) created by this delta.

-m[mrlist]

If the SCCS file has the *v* flag set, see *admin*, then a modification request (MR) number must be supplied as the reason for creating the new delta.

If *-m* is not used and the standard input is a terminal, the prompt "*MRs?*" is issued on the standard output before the standard input is read; if the standard

input is not a terminal, no prompt is issued. The “MRs?” prompt always precedes the “comments?” prompt (see *-y* option).

MRs in a list are separated by blanks and/or tab characters. An unescaped newline character terminates the MR list.

Note that if the *v* flag has a value, it is taken to be the name of a program which will validate the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, *delta* terminates. (It is assumed that the MR numbers were not all valid.)

-y[comment]

Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If *-y* is not specified and the standard input is a terminal, the prompt “comments?” is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped newline character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in *diff* format, see *diff*.

APPLICATION USAGE

Some systems limit the size of text to *-y[comment]*; if present, this limit will not be less than 512 bytes.

SEE ALSO

admin, diff, get, prs, rmdel.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

df – report free disk space

SYNOPSIS

OF UN **df** [**-t**][**file-system ...**]

DESCRIPTION

The *df* utility prints out the free space (in 512-byte units) and the number of free file slots, or *inodes*, available, for file systems. The argument *file-system* may be specified either by device name (e.g., **/dev/dsk/0s1**) or by mounted directory name (e.g., **/usr**). If no *file-system* is specified, the free space on all of the mounted file systems is printed.

OF UN The **-t** option causes the total allocated-space figures to be reported as well.

APPLICATION USAGE

Currently not all systems report in terms of 512-byte units. This situation may change in the future.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

diff – differential file comparator

SYNOPSIS

PI **diff [-efbh] file1 file2**

DESCRIPTION

The *diff* utility indicates those lines that must be changed in two files to bring them into agreement. If *file1* (*file2*) is *-*, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 a n3,n4

n1,n2 d n3

n1,n2 c n3,n4

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging *a* for *d* and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines, are all the lines that are affected in the first file flagged by *<*, then all the lines that are affected in the second file flagged by *>*.

The *-b* option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The *-e* option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*.

The *-f* option produces a similar script, not useful with *ed*, in the opposite order.

PI Option *-h* does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

PI Options *-e* and *-f* are unavailable with the *-h* option.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXIT STATUS

Exit status is:

- 0 no differences
- 1 differences
- 2 errors

APPLICATION USAGE

Editing scripts produced under the *-e* or *-f* option may be incorrect when dealing with lines consisting of a single period.

SEE ALSO

cmp, comm, ed.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

dircmp – directory comparison

SYNOPSIS

OF **dircmp** [-d] [-s] dir1 dir2

DESCRIPTION

The *dircmp* utility examines the directory hierarchies specified by *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is specified, a list is output that indicates whether the filenames common to both directories have the same contents.

- d Compare the contents of files with the same name in both directories and output a list indicating what must be changed in the two files to bring them into agreement. The list format is described in *diff*.
- s Suppress messages about identical files.

SEE ALSO

cmp, *diff*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

dis – disassembler (DEVELOPMENT) (OPTIONAL)

SYNOPSIS

PI `dis [-o] [-V] [-L] [-F function] [-l string] file ...`

DESCRIPTION

The *dis* utility produces an assembly language listing of each of its *file* arguments, each of which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following options are interpreted by the disassembler and may be specified in any order.

- o Will print numbers in octal. Default is hexadecimal.
- MV -V Version number of the disassembler will be written to standard error.
- L Invoke a lookup of C-language source labels in the symbol table for subsequent printing.
- F *function*
Disassemble only the named *function* in each object file specified on the command line. This option may be specified a number of times on the command line.
- l *string*
Will disassemble the library file specified as *string*. For example, the command *dis -lm* will disassemble the maths library.

FUTURE DIRECTIONS

The *-s* option is reserved for future use. It will be used to specify symbolic disassembly.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

du – estimate file-space usage

SYNOPSIS

OF PT **du** [**-ars**] [**file ...**]

DESCRIPTION

The *du* utility gives an estimate, in 512-byte units, of the file space contained in all the specified *files*. Whenever a directory is named, all files within it are reported; subdirectories are traversed recursively. If no *file* is specified, the current directory is used.

The option **-s** causes only the grand total (for each of the specified *files*) to be given. The option **-a** causes a report to be generated for each *file*. With no options, a report is given for each directory only.

UN Some implementations of *du* are silent about directories that cannot be read, files that cannot be opened, etc. If this is the case, the **-r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

APPLICATION USAGE

If the **-a** option is not used, non-directories given as arguments are not listed.

Files with holes in them may get an incorrect (high) estimate.

Currently not all systems report in terms of 512-byte units. This situation may change in the future.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

echo – echo arguments

SYNOPSIS

echo [**arg ...**]

DESCRIPTION

The *echo* command writes its *arguments* separated by blanks and terminated by a newline on the standard output. It also understands the following escape conventions:

\b	backspace
\c	print arguments up to this point, without newline; ignore remainder of command line
\f	form-feed
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\0n	<i>n</i> must be a 1-, 2- or 3-digit octal number; specifies the corresponding character

This command operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

The *echo* command is useful for producing diagnostics in command scripts and for sending known data into a pipe.

Arguments containing blanks and escape sequences must be suitably quoted.

This is usually a shell built-in command.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

ed, red – text editor

SYNOPSIS

ed [-] [-p string] [file]

red [-] [-p string] [file]

DESCRIPTION

The *ed* utility is a text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into the *ed* buffer so that it can be edited.

The *-* option suppresses the printing of byte counts by *e*, *r* and *w* commands, of diagnostics from *e* and *q* commands and of the *!* prompt after a *!command*.

The *-p* option allows the user to specify a prompt string.

The *ed* utility operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

The *red* utility is a restricted version of *ed*. It will only allow editing of files in the current directory and prohibits executing commands via *!command*. Attempts to bypass these restrictions result in an error message.

Commands to *ed* have a simple and regular structure: zero, one or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands are recognised; all input is merely collected. Input mode is left by typing a period, ".", alone at the beginning of a line.

IN The *ed* utility supports simple regular expression syntax. On systems supporting an internationalised environment, simple internationalised regular expression syntax is used (see Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions).

To understand addressing in *ed* it is necessary to know that at any time there is a current line. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. Addresses are constructed as follows:

1. The character "." addresses the current line.
2. The character \$ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*th line of the buffer.
4. The 'x' command addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines may be marked using the *k* command,

described below.

5. A regular expression (RE) enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. The null RE (i.e., / /) is equivalent to the last RE encountered.
6. An RE enclosed in question marks, ?, addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line.
7. An address followed by a plus sign, +, or a minus sign, -, followed by a decimal number specifies that address plus (or minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean . -5 .
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma "," stands for the address pair 1, \$, while a semicolon ";", stands for the address pair ., \$.

Commands may require zero, one or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma, ",". They may also be separated by a semicolon, ";". In the latter case, the current line, ".", is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally wrong for more than one command to appear on a line. However, any command (except *e*, *f*, *r* or *w*) may be suffixed by *l*, *n* or *p* in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a

<text> The append command reads the given *text* and appends it after the addressed line; the current line becomes the last inserted line, or, if there were none, the addressed line. Address 0 is valid for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of bytes that may be entered from a terminal is 256 per line (including the newline character).

(.)c

<text> The change command deletes the addressed lines, then accepts input text that replaces these lines; "." is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file* The edit command causes the entire contents of the buffer to be deleted, and then the named *file* to be read in; "." is set to the last line of the buffer. If no filename is given, the currently remembered filename, if any, is used (see the *f* command). The number of bytes read is typed; the name *file* is remembered for possible use as a default filename in subsequent *e*, *r* and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a command to the command interpreter whose output is to be read. Such a command is not remembered as the current filename.

E *file* The E (edit) command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file* If *file* is given, the filename command changes the currently remembered filename to *file*; otherwise, it prints the currently remembered filename.

(1,\$)g/RE/command list

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with "." initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i* and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v* and *V* commands are not permitted in the *command list*.

(1,\$)G/RE/

In the interactive global command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, "." is changed to that line and any one command (other than one of the *a*, *c*, *i*, *g*, *G*, *v* and *V* commands) may be input and executed. After the execution of that command, the next marked line is printed and so on; a newline acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the

execution of the *G* command may address and affect any lines in the buffer. The *G* command can be terminated by sending an interrupt signal.

LA **h** The help command gives a short error message that explains the reason for the most recent ? diagnostic.

LA **H** The help command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)**i**

<text> The insert command inserts the given *text* before the addressed line; "." is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not valid for this command. The maximum number of bytes that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)**j**

The join command joins contiguous lines by removing the appropriate newline characters. If exactly one address is given, this command does nothing.

(.)**kx**

The mark command marks the addressed line with name *x*, which must be a lower-case letter. The address *ˆx* then addresses this line; "." is unchanged.

OF (.,.)**l**

The list command prints the addressed lines in an unambiguous way. Non-printing characters are printed in a (hopefully) mnemonic form, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r* or *w*.

(.,.)**ma**

The move command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; "." is left at the last line moved.

(.,.)**n**

The number command prints the addressed lines, preceding each line by its line number and a tab character; "." is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r* or *w*.

(.,.)**p**

The print command prints the addressed lines; "." is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r* or *w*. For example, *dp* deletes the current line and prints the new current line.

P The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

- q The quit command causes *ed* to exit.
- Q The editor exits without checking if changes have been made in the buffer since the last *w* command.

(*\$*)*r file*

The read command reads in the given *file* after the addressed line. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands). The currently remembered filename is not changed unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of bytes read is typed; "." is set to the last line read in. If *file* is replaced by *!*, the rest of the line is taken to be a command to the command interpreter whose output is to be read. Such a command is not remembered as the current filename.

(*.,.*)*s*/*RE*/*replacement*/ or
 (*.,.*)*s*/*RE*/*replacement*/*g* or
 (*.,.*)*s*/*RE*/*replacement*/*n* n = 1-512

The substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n* appears after the command, only the *n*th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline may be used instead of */* to delimit the RE and the *replacement*; "." is left at the last line on which a substitution occurred.

An ampersand, *&*, appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of *&* in this context may be suppressed by preceding it by **. As a more general feature, the characters *\n*, where *n* is a digit, are replaced by the text matched by the *n*th regular subexpression of the specified RE enclosed between *\(* and *\)*. When nested parenthesised subexpressions are present, *n* is determined by counting occurrences of *\(* starting from the left. When the character *%* is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The *%* loses its special meaning when it is in a *replacement* string of more than one character or is preceded by a **.

A line may be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it by **. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)*ta*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); "." is left at the last line of the copy.

u The undo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G* or *V* command.

(1,\$)*v*/RE/*command list*

This command is the same as the global command *g* except that the *command list* is executed with "." initially set to every line that does not match the RE.

(1,\$)*V*/RE/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do not match the RE.

(1,\$)*w* *file*

The write command writes the addressed lines into the named file. The currently remembered filename is not changed unless *file* is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands); "." is unchanged. If the command is successful, the number of bytes written is typed. The file is created if it does not exist. If *file* is replaced by !, the rest of the line is taken to be a command to the command interpreter whose standard input is the addressed lines. Such a command is not remembered as the current filename.

(\$)=

The line number of the addressed line is typed; "." is unchanged by this command.

!*command*

The remainder of the line after the ! is sent to the command interpreter to be interpreted as a command. Within the text of that *command*, the unescaped character % is replaced with the remembered filename; if a ! appears as the first character of the command, it is replaced with the text of the previous command. Thus, !! will repeat the last command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)

An address alone on a line causes the addressed line to be printed. A newline alone is equivalent to .+1p; it is useful for stepping forward through the buffer.

If an interrupt signal is sent, *ed* prints a ? and returns to its command level.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a newline, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2    s/s1/s2/p
g/s1       g/s1/p
?s1        ?s1?p
```

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy the editor buffer via the *e* or *q* commands. It prints ? and allows the user to continue editing. A second *e* or *q* command at this point will take effect. The - command line option inhibits this feature.

If *ed* receives a SIGHUP signal, work is saved in a file named **ed.hup**.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of range expressions, equivalence classes and multi-character collating symbols, and *LC_CTYPE* determines the behaviour of character classes.

LC_CTYPE also determines which characters are classified as non-printing for the *l* (list) command.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

ERRORS

? for command errors.
?file for an inaccessible file.

Use the *h* and *H* (help) commands for detailed explanations.

APPLICATION USAGE

A ! command cannot be subject to a *g* or a *v* command.

The sequence \n in an RE does not match a newline character.

If the editor input is coming from a *file* command (e.g., *ed file <ed-cmd-file*), the editor will exit at the first failure of a command that is in the command file.

FUTURE DIRECTIONS

The option - will be replaced by -s, in order to conform to the syntax standard. The old form of the option will continue to be accepted for some time.

SEE ALSO

sed, sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an international environment has been described.

NAME

egrep, fgrep – search a file for a pattern

SYNOPSIS

LV **egrep** [options] [expression] [file ...]

LV **fgrep** [options] [string ...] [file ...]

DESCRIPTION

The *egrep* and *fgrep* utilities search the input *file* or files (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output.

IN The patterns used by the *egrep* utility make use of extended regular expression syntax. On systems supporting an internationalised environment, extended internationalised regular expression syntax is used (see **Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions**). The patterns used by the *fgrep* utility are fixed strings. The following *options* are recognised:

- v All lines but those matching are printed.
- x (Exact) Only lines matched in their entirety are printed (*fgrep* only).
- c Only a count of matching lines is printed.
- i Ignore upper- and lower-case distinction during comparisons.
- l Only the names of files with matching lines are listed (once), separated by newlines.
- n Each line is preceded by its relative line number in the file.
- e *expression*
(*Egrep* only.) Same as a simple expression argument, but useful when the *expression* begins with –.

–f *file*

The regular expression (*egrep*) or strings list (*fgrep*) is taken from the *file*.

In all cases, the filename is **output** if there is more than one input file. Care should be taken when using characters in *expression* that may also be meaningful to the command interpreter. When using *sh*, it is safest to enclose the entire *expression* argument in single quotes ‘...’.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

fgrep

The *fgrep* utility searches for lines that contain one of the *strings* separated by newlines.

egrep

IN The *egrep* utility makes use of extended regular expression syntax, with the additional feature that newline is also a regular expression alternation character. On systems supporting an internationalised environment, extended internationalised regular expression syntax is used (see **Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions**).

Environment Variables

IN (Egrep and fgrep). *LC_CTYPE* determines which characters are defined as letters (character class *alpha*) and identifies shift information for the *-i* option.

(Egrep only). Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

Exit status is:

- 0 matches are found.
- 1 no matches found.
- 2 syntax errors or inaccessible files (even if matches found in other files).

APPLICATION USAGE

Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in *<stdio.h>*.)

FUTURE DIRECTIONS

The functionality of *egrep* and *fgrep* will eventually be provided in *grep* and these two utilities discontinued.

SEE ALSO

grep, *sed*, *<stdio.h>*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions used by *egrep* have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

env – set environment for command execution

SYNOPSIS

env [-] [name=value...] [command [arg...]]

DESCRIPTION

The *env* utility obtains the current environment, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* modify the execution environment: they are merged into the inherited environment before the command is executed. The *-* option causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

Optionally, arguments may be passed to the command by placing them as separate words after the command name.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

ex – text editor

SYNOPSIS

OP **ex** [-] [-v] [-r] [-R] [+command] [-l] [file ...]

DESCRIPTION

The *ex* utility is a line oriented text editor, which supports both command and display editing, see *vi*. The command line options are:

- Suppress all interactive-user feedback. This is useful in processing editor scripts.
- v Invoke *vi*.
- r Recover *file* or files after an editor or system crash. If no *file* is specified a list of all saved files will be printed.
- R Set read-only mode; prevents accidentally overwriting the file.
- +command
Begin editing by executing the specified editor search or positioning *command*.
- l Set lisp mode; indents appropriately for Lisp code; the (), {}, [[and]] commands in *vi* are modified to have meaning for Lisp.

The *file* argument(s) indicates files to be edited, in the order specified.

The name of the file being edited by *ex* is the *current* file. The text of the file is read into a buffer and all editing changes are performed in this buffer; changes have no effect on the file until the buffer is written out explicitly.

The *alternate* filename is the name of the last file mentioned in an editor command, or the previous current filename if the last file mentioned became the current file. The character % in filenames is replaced by the current filename, and the character # by the alternate filename.

The named buffers *a* to *z* may be used for saving blocks of text during the edit. If the buffer name is specified in upper-case, the buffer is appended to rather than being overwritten.

The read-only mode can be cleared from within the edit by setting the no-read-only edit option. Writing to a different file is allowed in read-only mode; in addition, the write can be forced by using ! (see the *write* command below).

When an error occurs *ex* prints a message. If an interrupt signal is received, *ex* returns to the command level in addition to the above actions. If the editor input is from a file, *ex* exits at the interrupt.

If *ex* detects an internal error, it attempts to preserve the buffer if any unwritten changes were made. The command line option -r is used to retrieve the saved changes.

At the beginning, *ex* is in the command mode, which is indicated by the ":" prompt. The input mode is entered by *append*, *insert*, or *change* commands; it is left (and

command mode re-entered) by typing "." alone at the beginning of a line.

Command lines beginning with the double quote character, ", are ignored. (This may be used for comments in an editor script.)

Addressing

Dot, ".", refers to the current line. There is always a current line; the positioning may be the result of an explicit movement by the user, or the result of a command that affected multiple lines (in which case it is usually the last line affected).

n The *n*th line in the buffer, with lines numbered sequentially from 1.

\$ The last line in the buffer.

% Abbreviation for 1,\$; the entire buffer.

+*n*

-*n* An offset relative to the current line. (The forms .+3, +3, and +++ are equivalent.)

/re/

?re? Line containing the pattern (see **Regular Expressions**, below) *re*, scanning forwards, //, or backwards, ??. The trailing / or ? may be omitted if the line is only to be printed. If the pattern is omitted, the previous pattern specified is used.

`x Lines may be marked using single lower-case letters (see the *mark* command below); `x refers to line marked *x*. In addition, the previous current line is marked before each non-relative motion; this line may be referred to by using ` for *x*.

Addresses to commands consist of a series of line addresses (specified as above), separated by ";" or ":". Such address lists are evaluated left to right. When ";" is the separator, the current line is set to the value of the previous address before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. Where a command requires two addresses, the first line must precede the second one in the buffer. A null address in a list defaults to the current line.

Command Names and Abbreviations

abbrev	<i>ab</i>	next	<i>n</i>	unmap	<i>unm</i>
append	<i>a</i>	number	<i># nu</i>	version	<i>ve</i>
args	<i>ar</i>	preserve	<i>pre</i>	visual	<i>vi</i>
change	<i>c</i>	print	<i>p</i>	write	<i>w</i>
copy	<i>co</i>	put	<i>pu</i>	xit	<i>x</i>
delete	<i>d</i>	quit	<i>q</i>	yank	<i>ya</i>
edit	<i>e</i>	read	<i>r</i>	(window)	<i>z</i>
file	<i>f</i>	recover	<i>rec</i>	(escape)	<i>!</i>
global	<i>g v</i>	rewind	<i>rew</i>	(lshift)	<i><</i>
insert	<i>i</i>	set	<i>se</i>	(rshift)	<i>></i>
join	<i>j</i>	shell	<i>sh</i>	(resubst)	<i>& s</i>
list	<i>l</i>	source	<i>so</i>	(scroll)	<i>^D</i>
map		substitute	<i>s</i>	(line no)	<i>=</i>
mark	<i>k ma</i>	unabbrev	<i>una</i>		
move	<i>m</i>	undo	<i>u</i>		

Command Descriptions

In the following, *line* is a single line address, given in any of the forms described in the **Addressing** section above; *range* is a pair of line addresses, separated by a comma or semicolon (see the **Addressing** section for the difference between the two); *count* is a positive integer, specifying the number of lines to be affected by the command; *flags* is one or more of the characters *#*, *p*, and *l*; the corresponding command to print the line is executed after the command completes. Any number of *+* or *-* characters may also be given with these flags.

When *count* is used, *range* is not effective; only a line number should be specified instead, to indicate the first line affected by the command. (If a range is given, then the last line of the range is taken as the starting line for the command.)

These modifiers are all optional; the defaults are as follows, unless otherwise stated: the default for *line* is the current line; the default for *range* is the current line only (*.,.*); the default for *count* is 1; the default for *flags* is null.

When only a *line* or a *range* is specified (with a null command), the implied command is *print*; if a null line is entered, the next line is printed (equivalent to *+.1p*)

ab *word rhs*

Add the named abbreviation to the current list. In visual mode, if *word* is typed as a complete word during input, it is replaced by the string *rhs*.

line a Enter input mode; the input text is placed after the specified line. If line 0 is specified, the text is placed at the beginning of the buffer. The last input line becomes the current line, or the target line, if no lines are input.

ar The argument list is printed, with the current argument inside [and].

range c count

Enter input mode; the input text replaces the specified lines. The last input line becomes the current line; if no lines are input, the effect is the same as a delete.

range co line flags

A copy of the specified lines (*range*) is placed after the specified destination line; line 0 specifies that the lines are to be placed at the beginning of the buffer.

range d buffer count

The specified lines are deleted from the buffer. If a named *buffer* is specified, the deleted text is saved in it. The line after the deleted lines becomes the current line, or the last line if the deleted lines were at the end.

e [+line] file

Begin editing a new file. If the current buffer has been modified since the last write, then a warning is printed and the command is aborted. This action may be overridden by appending the character ! to the command (*e! file*). The current line is the last line of the buffer; however, if this command is executed from within *visual*, the current line is the first line of the buffer. If the *+line* option is specified, the current line is set to the specified position, where *line* may be a number (or \$) or specified as */re* or *?re*.

f

Print the current filename and other information, including the number of lines and the current position.

range g /re/cmds ...

First marks the lines within the given *range* that match the given pattern. Then the given *command* or commands are executed with "." set to each marked line.

The *cmds* may be specified on multiple lines by hiding newlines with a backslash. If *cmds* are omitted, each line is printed. *Append*, *change*, and *insert* commands are allowed; the terminating dot may be omitted if it ends *cmds*. The *visual* command is also permitted, and takes input from the terminal.

The *global* command itself, and the *undo* command, are not allowed in *cmds*. The edit options *autoprint*, *autoindent* and *report* are inhibited.

range v /rel/cmds ...

This is the same as the *global* command, except that *cmds* is run on the lines that do not match the pattern.

line i

Enter input mode; the input text is placed before the specified line. The last line input becomes the current line, or the line before the target line, if no lines were input.

range j count flags

Join the text from the specified lines together into one line. White space is adjusted to provide at least one blank character, two if there was a period at the end of the line, or none if the first following character is "). Extra white space at the start of a line is discarded.

Appending the command with ! causes a simpler join with no white space processing.

range l count flags

Print the specified lines with tabs printed as ^I and the end of each line marked

with a trailing \$. (The only useful flag is # for line numbers.) The last line printed becomes the current line.

map *x rhs*

The *map* command is used to define macros for use in visual mode. The first argument is a single character, or the sequence #*n*, where *n* is a digit, to refer to the function key *n*. When this character or function key is typed in visual mode, the action is as if the corresponding *rhs* had been typed. If ! is appended to the *map* command, then the mapping is effective during insert mode rather than command mode. Special characters, white space, and newline must be escaped with a ^*V* to be entered in the arguments.

line ma *x*

(The letter *k* is an alternative abbreviation for the *mark* command.) The specified line is given the specified mark *x*, which must be a single lower-case letter. (The *x* must be preceded by a space or tab.) The current line position is not affected.

range m *line*

Move the specified lines (*range*) to be after the target line. The first of the moved lines becomes the current line.

n The next file from the command line argument list is edited. Appending ! to the command overrides the warning about the buffer having been modified since the last write (discarding any changes). The argument list may be replaced by specifying a new one on this command line.

range nu *count flags*

(The character # is an alternative abbreviation for the *number* command.) Print the lines, each preceded by its line number. (The only useful flag is !.) The last line printed becomes the current line.

pre The current editor buffer is saved as though the system had just crashed. This command is for use in emergencies, for example when a write does not work, and the buffer cannot be saved in any other way.

range p *count*

Prints the specified lines, with non-printing characters printed as control characters in the form ^*x*; DEL is represented as ^?. The last line printed becomes the current line.

line pu *buffer*

Put back deleted or "yanked" lines. A buffer may be specified; otherwise, the text in the unnamed buffer (where deleted or yanked text is placed by default) is restored.

q Causes termination of the edit. If the buffer has been modified since the last write, a warning is printed and the command fails. This warning may be overridden by appending ! to the command (discarding changes).

line r *file*

Place a copy of the specified *file* in the buffer after the target line (which may be

line 0 to place text at the beginning). If no *file* is named the current file is the default. If there is no current file then *file* becomes the current file. The last line read becomes the current line; in *visual* the first line read becomes the current line.

If *file* is given as *!string* then *string* is taken to be a system command, and passed to the command interpreter; the resultant output is read in to the buffer. A blank or tab must precede !.

rec *file*

Recovers *file* from the save area, after an accidental hangup or a system crash.

rew The argument list is rewound, and the first file in the list is edited. Any warnings may be overridden by appending !.

se *parameter*

With no arguments, the *set* command prints those options whose values have been changed from the default settings; with the parameter *all* it prints all of the option values.

Giving an option name followed by ? causes the current value of that option to be printed. The ? is necessary only for Boolean-valued options. Boolean options are given values by the form *se option* to turn them on, or *se no option* to turn them off; string and numeric options are assigned by the form *se option=value*. More than one parameter may be given; they are interpreted left to right.

See **Edit Options** below for further details about options.

sh The user is put into the command interpreter, usually *sh*, see *sh*; editing is resumed on exit.

so *file*

Read and execute commands from the specified *file*. Such *so* commands may be nested.

range s /re/repl/ options count flags

On each specified line, the first instance of the pattern *re* is replaced by the string *repl*. (See **Regular Expressions and Replacement Strings** below.) If *options* includes the letter *g* (global), then all instances of the pattern in the line are substituted. If the option letter *c* (confirm) is included, then before each substitution the line is typed with the pattern to be replaced marked with ^ characters; a response of *y* causes the substitution to be done, while any other input aborts it. The last line substituted becomes the current line.

una *word*

Delete *word* from the list of abbreviations.

u Reverse the changes made by the previous editing command. For this purpose, *global* and *visual* are considered single commands. Commands which affect the external environment, such as *write*, *edit* and *next*, cannot be undone. An *undo* can itself be reversed.

unm *x*

The macro definition for *x* is removed.

MV **ve** Print the current version of the editor.

line vi type count

Enter visual mode at the specified *line*. The *type* is optional, and may be – or “.”, as in the *z* command, to specify the position of the specified line on the screen window. (The default is to place the line at the top of the screen window.) A *count* specifies an initial window size; the default is the value of the *window* option. The *Q* command exits visual mode. For more information, see *vi*.

range w file

Write the specified lines (the whole buffer, if no *range* is given) out to *file*, printing the number of lines and characters written. If *file* is not specified, the default is the current file. (The command fails with an error message if there is no current file and no file is specified.)

If an alternate file is specified, and the file exists, then the write will fail; it may be forced by appending a *!* to the command. An existing file may be appended to by appending *>>* to the command. If the file does not exist, an error is reported.

If the file is specified as *!string*, then *string* is taken as a system command; the command interpreter is invoked, and the specified lines are passed as standard input to the command.

The *wq* command is equivalent to a *w* followed by a *q*; *wq!* is equivalent to *w!* followed by *q*.

x Write out the buffer if any changes have been made, and then (in any case) quits.

range ya buffer count

Place the specified lines in the named *buffer*. If no buffer is specified, the unnamed buffer is used (where the most recently deleted or yanked text is placed by default).

line z type count

If *type* is omitted, then *count* lines following the specified *line* (default current line) are printed. The default for *count* is the value of the *window* option.

If *type* is specified, it must be – or “.”; a – causes the line to be placed at the bottom of the screen, while a “.” causes the line to be placed in the middle. The last line printed becomes the current line.

! command

The remainder of the line after the *!* is passed to the system command interpreter for execution. A warning is issued if the buffer has been changed since the last write. A single *!* is printed when the command completes. The current line position is not affected.

Within the text of *command*, % and # are expanded as filenames, and ! is replaced with the text of the previous ! command. (Thus !! repeats the previous ! command.) If any such expansion is done, the expanded line will be echoed.

range! *command*

In this form of the ! command, the specified lines (there is no default; see previous paragraph) are passed to the command interpreter as standard input; the resulting output replaces the specified lines.

range < *count*

Shift the specified lines to the left; the number of spaces to be shifted is determined by the edit *shiftwidth* option. Only white space (blanks and tabs) is lost in shifting; other characters are not affected. The last line changed becomes the current line.

range > *count*

Shift the specified lines to the right by inserting white space (see previous paragraph for further details).

range & *options count flags*

Repeat the previous *substitute* command, as if & were replaced by the previous *s/re/repl/*. (The same effect is obtained by omitting the */re/repl/* string in the *substitute* command.)

^D (control-D)

^D (ASCII EOT) prints the next *n* lines, where *n* is the value of the *scroll* option.

line =

Print the line number of the specified *line* (default last line). The current line position is not affected.

Regular Expressions

Regular expressions are interpreted according to the setting of the *magic* option; the following assumes the setting *magic*. The differences caused by the setting *nomagic* are described below.

The following constructs are used to construct regular expressions:

char An ordinary character matches itself. The following characters are not ordinary, and must be escaped (preceded) by \ to retain their ordinary meaning: ^ at the beginning of a pattern, \$ at the end of a pattern, * anywhere other than the beginning of a pattern, ".", [and ~ anywhere in a pattern.

^ When it appears at the beginning of a pattern, ^ matches the beginning of the line.

\$ When it appears at the end of a pattern, \$ matches the end of the line.

.

The "." matches any single character in the line.

\< The \< matches the beginning of a "word"; that is, the matched string must begin in a letter, digit, or underline, and be preceded by the beginning of the line or a character other than the above.

\> The \> matches the end of a "word" (see previous paragraph).

[string]

Match any single character in string. Within string, the following have special meanings: A pair of characters separated by "--" defines a range (for example, [a-z] defines any lowercase letter in the ASCII code set); the character ^, if it is the first character in string, causes the construct to match characters other than those specified in string. These special meanings can be escaped by preceding them with a backslash, \.

* Matches zero or more occurrences of the preceding regular expression.

~ Matches the replacement part of the last substitute command.

\(pattern\)

A regular expression pattern can be enclosed in escaped parentheses; this serves only to identify them for substitution actions (see **Replacement Strings** below).

A concatenation of two regular expressions is a regular expression that matches the concatenation of the strings matched by each component.

When *nomagic* is set, the only characters with special meanings are ^ at the beginning of a pattern, \$ at the end of a pattern, and \. The characters ".", *, [, and ~ lose their special meanings, unless escaped by a \.

Replacement Strings

The character & (\& if *nomagic* is set) in the replacement string stands for the text matched by the pattern to be replaced. The character ~ (\~ if *nomagic* is set) is replaced by the replacement part of the previous *substitute* command. The sequence \n, where n is an integer, is replaced by the text matched by the pattern enclosed in the nth set of parentheses \(and \). The sequence \u (\l) causes the immediately following character in the replacement to be converted to upper-case (lower-case), if this character is a letter. The sequence \U (\L) turns such conversion on, until the sequence \E or \e is encountered, or the end of the replacement string is reached.

Edit Options

The *ex* utility has a number of options that modify its behaviour. These options have default settings, which may be changed using the *set* command (see above). Options may also be set at start-up by putting a *set* command string in the environment variable *EXINIT*, or in the file *.exrc* in the *HOME* directory, or in *.exrc* in the current directory.

Options are Boolean unless otherwise specified.

autoindent, ai

If autoindent is set, each line in insert mode is indented (using blanks and tabs) to align with the previous line. (Starting indentation is determined by the line appended after, or the line inserted before, or the first line changed.) Additional indentation can be provided as usual; succeeding lines will

automatically be indented to the new alignment. Reducing the indent is achieved by typing `^D` one or more times; the cursor is moved back *shiftwidth* spaces for each `^D`. (A `^` followed by a `^D` removes all indentation temporarily for the current line; a `0` followed by a `^D` removes all indentation.)

autoprint, ap

The current line is printed after each command that changes buffer text. (Autoprint is suppressed in globals.)

autowrite, aw

The buffer is written (to the current file) if it has been modified, and a *next*, *rewind*, or `!` command is given.

beautify, bf

Causes all control characters other than *tab*, *newline* and *form-feed* to be discarded from the input text.

directory, dir

The value of this option specifies the directory in which the editor buffer is to be placed. If this directory is not writable by the user, the editor quits.

edcompatible, ed

Causes the presence of *g* and *c* suffixes on substitute commands to be remembered, and toggled by repeating the suffixes.

ignorecase, ic

All upper-case characters in the text are mapped to lower-case in regular expression matching. Also, all upper-case characters in regular expressions are mapped to lower-case, except in character class specifications.

lisp Autoindent mode, and the `(`, `)`, `{`, `}`, `[[` and `]]` commands in *visual* are suitably modified for *lisp* code.

list All printed lines will be displayed with tabs shown as `^I`, and the end of line marked by a `$`.

magic

Changes interpretation of characters in regular expressions and substitution replacement strings (see the relevant sections above).

number, nu

Causes lines to be printed with line numbers.

paragraphs, para

The value of this option is a string, in which successive pairs of characters specify the names of text-processing macros which begin paragraphs. (A macro appears in the text in the form `.XX`, where the `"."` is the first character in the line.)

prompt

When set, command mode input is prompted for with a `":"`; when unset, no prompt is displayed.

redraw

The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

remap

If set, then macro translation allows for macros defined in terms of other macros; translation continues until the final product is obtained. If unset, then a one-step translation only is done.

report

The value of this option gives the number of lines that must be changed by a command before a report is generated on the number of lines affected.

scroll

The value of this option determines the number of lines scrolled on a `^D`, and the number of lines displayed by the `z` command (twice the value of scroll).

sections

The value of this option is a string, in which successive pairs of characters specify the names of text-processing macros which begin sections. (See *paragraphs* option above.)

shiftwidth, sw

The value of this option gives the width of a software tab stop, used during *autoindent*, and by the shift commands.

showmatch, sm

In visual mode, when a `)` or `}` is typed, the matching `(` or `{` is shown if it is still on the screen.

slowopen, slow

In visual mode, prevents screen updates during input to improve throughput on unintelligent terminals.

tabstop, ts

The value of this option specifies the software tab stops to be used by the editor to expand tabs in the input file.

terse

When set, error messages are shorter.

window

The number of lines in a text window in visual mode.

wrapscan, ws

When set, searches (using `//` or `??`) wrap around the end of the file; when unset, searches stop at the beginning or the end of the file, as appropriate.

wrapmargin, wm

In visual mode, if the value of this option is greater than zero (say *n*), then a newline is automatically added to an input line, at a word boundary, so that lines end at least *n* spaces from the right margin of the terminal screen.

writeany, wa

Inhibits the checks otherwise made before write commands, allowing a write to any file (provided the system allows it).

APPLICATION USAGE

The *undo* command causes all marks to be lost on lines that were changed and then restored.

The *z* command prints a number of logical rather than physical lines. More than a screenful of output may result if long lines are present.

Null characters are discarded in input files and cannot appear in resultant files.

On some systems, recovery of an edit lost due to a system crash may only be possible if certain system-dependent actions were taken when the system was restarted.

As this editor depends on the optional *Curses* package, (see **Volume 3, XSI Curses Interface**), applications should be aware that it may not be present, or may not function with certain classes of terminal.

SEE ALSO

vi.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The `""` regular expression notation has been correctly documented.

NAME

expr – evaluate expression

SYNOPSIS

expr expression

DESCRIPTION

The *expr* utility evaluates *expression* and writes the result on the standard output. Terms of the expression must be separated by blanks. Characters special to the command interpreter must be suitably escaped. Note that 0 is returned to indicate a zero value, rather than the empty string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign.

The operators are listed below. The list is in order of increasing precedence, with equal precedence operators grouped within {} symbols. The symbol *arg* represents an argument.

arg | *arg*

return the first *arg* if it is neither null nor 0, otherwise return the second *arg*.

arg & *arg*

return the first *arg* if neither *arg* is null nor 0, otherwise return 0.

arg { =, >, >=, <, <=, != } *arg*

return the result of an integer comparison if both arguments are integers, otherwise return the result of a lexical comparison. Return an exit status of 0 for false and 1 for true.

arg { +, - } *arg*

Addition or subtraction of integer-valued arguments.

arg { *, /, % } *arg*

Multiplication, division, or remainder of the integer-valued arguments.

arg : *arg*

IN

The matching operator, ":", compares the first argument with the second argument which must be a regular expression. Simple regular expression syntax is used except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. On systems supporting an internationalised environment, simple internationalised regular expression syntax is used (see Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions). Normally, the matching operator returns the number of bytes matched (0 on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

This utility operates in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

Environment Variables

IN Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes. *LC_COLLATE* determines the behaviour of the relational operators when comparing string values, e.g.,

```
expr $reply = "y"
```

where the value of *\$reply* might be *ÿ* and *LC_COLLATE* applies primary weight; only then *ÿ* and *y* would compare equally.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

As a side-effect of expression evaluation, *expr* returns the following exit values:

- 0 if the expression is neither null nor 0.
- 1 if the expression is null or 0.
- 2 for invalid expressions.

EXAMPLES

1. The following example adds 1 to the variable *a*.

```
a=`expr $a + 1`
```

2. For *\$a* equal to either */usr/abc/file* or just *file*, the following example returns the last segment of a pathname (i.e., *file*). Care should be taken when using */* alone as an argument: *expr* will take it as the division operator.

```
expr $a : '.*\/ (.*)' \| $a
```

3. A better representation of example 2: the addition of the *//* characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr // $a : '.*\/ (.*)'
```

4. The example below returns the number of bytes in *\$VAR*.

```
expr $VAR : '.*'
```

APPLICATION USAGE

After argument processing, see *sh*, *expr* cannot tell the difference between an operator and an operand except by the value. If *\$a* is *=*, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator).

The following works:

```
expr X$a = X=
```

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

find – find files

SYNOPSIS

find *pathname* ... *expression*

DESCRIPTION

The *find* utility recursively descends the directory hierarchy for each *pathname* in the *pathname* list (i.e., one or more pathnames) seeking files that match a Boolean *expression* written in the primaries given below. In the following descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *-n* means less than *n* and *n* means exactly *n*.

The *expression* argument is made up of:

-name *file*

True if *file* matches the current filename. Shell metanotation characters may be used in *file*. (See **Section 2.2, Shell Metanotation**).

-perm *onum*

True if the file permission flags exactly match the octal number *onum*, see *chmod*. If *onum* is prefixed by “-”, more flag bits (017777), see *stat()*, become significant and the flags are compared.

-type *c*

True if the type of the file is *c*, where *c* is *b*, *c*, *d*, *p* or *f* for block special file, character special file, directory, FIFO (named pipe) or plain file respectively. Other types may exist, but should be avoided if portability is an issue.

-links *n*

True if the file has *n* links.

-user *uname*

True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the User Database, it is taken as a user ID.

-group *gname*

True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the Group Database, it is taken as a group ID.

-size *n*[Pc]

True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a *c*, the size is in bytes.

-atime *n*

True if the file has been accessed in *n* days. The access time of directories in the *pathname* list is changed by *find* itself.

-mtime *n*

True if the file has been modified in *n* days.

-ctime *n*

True if the file has been changed in *n* days.

-exec *cmd*

True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped `“;”`. A command argument `{}` is replaced by the current pathname.

-ok *cmd*

IN

Like `-exec` except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing `y` or the locale's equivalent of `y`.

-print

Always true; causes the current pathname to be printed.

-newer *file*

True if the current file has been modified more recently than the argument *file*.

-depth

Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*, see *cpio*, to transfer files that are contained in directories without write permission.

(*expression*)

True if the parenthesised expression is true (parentheses must be escaped if they are special to the command interpreter).

The primaries may be combined using the following operators (in order of decreasing precedence):

1. The negation of a primary (`!` is the unary *not* operator).
2. Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3. Alternation of primaries (`-o` is the *or* operator).

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN

LANG determines the locale's equivalent of *y* (for *yes* / *no* queries). Within bracketed filename patterns, e.g.,

```
find . -name '[[=a=]]*' -print
```

LC_COLLATE determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXAMPLE

To remove all files named **tmp** or ending in **.xx** that have not been accessed for a week:

```
find / \ ( -name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;
```

SEE ALSO

chmod, *cpio*, *sh*, *test*, **Volume 2, XSI System Interfaces and Headers**, *stat()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

gencat – generate a formatted message catalogue

SYNOPSIS

gencat catfile msgfile ...

DESCRIPTION

The *gencat* utility merges the message text source file(s) *msgfile* into a formatted message catalogue *catfile*. The file *catfile* will be created if it does not already exist. If *catfile* does exist, its messages will be included in the new *catfile*. If set and message numbers collide, the new message text defined in *msgfile* will replace the old message text currently contained in *catfile*.

The format of message text source files is defined in **Volume 3, XSI Internationalisation, Section 5.2.1, Message Text Source Files**.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Message catalogues produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C-programs need to be recompiled for each type of machine, so message catalogues must be recreated via *gencat*.

FUTURE DIRECTIONS

The /usr/group and POSIX committees are still working on a complete definition of message labels. The definition in this Guide will be aligned with the eventual outcome of their work.

CHANGE HISTORY

First released in Issue 3.

NAME

get – get a version of an SCCS file (**DEVELOPMENT**)

SYNOPSIS

```
get [-rSID] [-ccutoff] [-e] [-b] [-ilist] [-xlist] [-k] [-l [p]] [-p] [-s] [-m] [-n]
    [-g] [-t] file ...
```

DESCRIPTION

The *get* utility generates an ASCII text file from each named SCCS *file* according to the specifications given by its options. The arguments may be specified in any order, but all options apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with *s.*) and unreadable files are silently ignored. If *-* is given as the name of a *file*, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS filename by simply removing the leading "*s.*".

Each of the options is explained below as though only one SCCS file is to be processed, but the effects of any option apply independently to each named file.

-rSID The SCCS Identification String (SID) of the version (delta) of an SCCS file to be retrieved. The table shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta* if the *-e* option is also used), as a function of the SID specified.

-ccutoff

The *cutoff* date-time, in the form:

```
YY[MM[DD[HH[MM[SS]]]]]
```

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; for example, *-c7502* is equivalent to *-c750228235959*.

Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: *-c77/2/2 9:22:25*.

-e Indicate that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*. The *-e* option used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the *j* (joint edit) flag is set in the SCCS file. Concurrent use of *get -e* for different SIDs is always allowed.

If the *g-file* generated by *get* with a *-e* option is accidentally ruined in the process of editing, it may be regenerated by re-executing the *get* command with the *-k* option in place of the *-e* option.

SCCS file protection specified via the ceiling, floor and authorised user list stored in the SCCS file is enforced when the *-e* option is used.

- b Use with the *-e* option to indicate that the new delta should have an SID in a new branch as shown in the table below. This option is ignored if the *b* flag is not present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.)

Note: A branch delta may always be created from a non-leaf delta.

- ilist A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of the table below. Partial SIDs are interpreted as shown in the "SID Retrieved" column of the table below.

- xlist A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the *-i* option for the *list* format.
- k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The *-k* option is implied by the *-e* option.
- l
- lp Cause a delta summary to be written into an *l-file*. If *-lp* is used, then an *l-file* is not created; the delta summary is written on the standard output instead.
- p Cause the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to standard error instead, unless the *-s* option is used, in which case it disappears.
- s Suppress all output normally written on the standard output. However, fatal error messages (which are always written to the standard error) remain unaffected.
- m Cause each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
- n Cause each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the *-m* and *-n* options are used, the format is: %M% value, followed by a horizontal tab, followed by the *-m* option generated format.

- g Suppress the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- t Use to access the most recently created ("top") delta in a given release (e.g., -r1), or release and level (e.g., -r1.2).

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the *-e* option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each filename is printed (preceded by a newline) before it is processed. If the *-i* option is used, included deltas are listed following the notation "Included"; if the *-x* option is used, excluded deltas are listed following the notation "Excluded".

Determination of SCCS Identification String				
SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk successor	R.L	R.(L+1)
R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	-	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch successor	R.L.B.S	R.L.(mB+1).1

- * R, L, B and S are the "release", "level", "branch" and "sequence" components of the SID, respectively; m means "maximum". Thus, for example, R.mL means "the maximum level number within release R"; R.L.(mB+1).1 means "the first sequence number on the new branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form R.L, R.L.B or R.L.B.S, each of the specified components must exist.

- ** hR is the highest existing release that is lower than the specified, non-existent, release R.
- *** This is used to force creation of the first delta in a new release.
- † The *-b* option is effective only if the *b* flag is present in the file. An entry of *-* means "irrelevant".
- ‡ This case applies if the *d* (default SID) flag is not present in the file. If the *d* flag is present in the file, then the SID obtained from the *d* flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

Identification Keywords

Identifying information is inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the <i>m</i> flag in the file, or if absent, the name of the SCCS file with the leading <i>s.</i> removed.
%I%	SCCS identification (SID) (%R%.%L% or %R%.%L%.%B%.%S%) of the retrieved text
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the <i>t</i> flag in the SCCS file.
%F%	SCCS filename.
%P%	Fully qualified SCCS filename.
%Q%	The value of the <i>q</i> flag in the file.
%C%	Current line number. This keyword is intended for identifying messages output by the program, such as "this should not have happened" type errors. It is not intended to be used on every line to provide sequence numbers.
%Z%	The four-character string @(#) recognisable by <i>what</i> .
%W%	A shorthand notation for constructing <i>what</i> strings. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing <i>what</i> strings. %A% = %Z%%Y%%M%%I%%Z%

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file* and *z-file*. The letter before the hyphen is called the *tag*. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*, the auxiliary files are named by replacing the leading *s* with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the *s.* prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c* and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *-p* option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the *-k* option is used or implied, it is writable by the owner only (read-only for everyone else); otherwise it is read-only. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the *-l* option is used; it is read-only and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied; * otherwise.
- b. A blank character if the delta was applied or was not applied and ignored; * if the delta was not applied and was not ignored.
- c. A code indicating a "special" reason why the delta was or was not applied:
 - I: Included.
 - X: Excluded.
 - C: Cut off (by a *-c* option).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with a *-e* option along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with a *-e* option for the same SID until *delta* is executed or the joint edit flag, *j*, is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. It is writable by owner only, and it is owned by the effective user. The format of the *p-file* is: the *g-file* SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the *-i* option if present, followed by a blank and the *-x* option if present, followed by a newline. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same

new delta SID.

The *z-file* serves as a lock-out mechanism against simultaneous updates. Its contents are the binary process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created read-only.

SEE ALSO

admin, delta, prs, what.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

grep – search a file for a pattern

SYNOPSIS

grep [options] expression [file ...]

DESCRIPTION

IN The *grep* utility searches the input *file* or files (standard input default) for lines matching the pattern *expression*. Normally, each line on which it matches is copied to the standard output. The patterns used by the *grep* utility make use of simple regular expression syntax. On systems supporting an internationalised environment, simple internationalised regular expression syntax is used (see Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions).

The following *options* are recognised:

- v All lines but those matching are printed.
- c Only a count of matching lines is printed.
- i Ignore upper- and lower-case distinction during comparisons.
- l Only the names of files with matching lines are listed (once), separated by newlines.
- n Each line is preceded by its relative line number in the file.
- s The error messages produced for non-existent or unreadable files are suppressed.

In all cases, the filename is output if there is more than one input file.

This utility operates in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

Environment Variables

IN Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes.

LC_CTYPE determines which characters are defined as letters (character class *alpha*) and identifies shift information for the *–i* option.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

Exit status is:

- 0 matches are found
- 1 none found
- 2 syntax errors or inaccessible files (even if matches found in other files)

APPLICATION USAGE

Care should be taken when using characters in *expression* that may also be meaningful to the command interpreter. It is safest to enclose the entire *expression* argument in single quotes: `'...'`.

FUTURE DIRECTIONS

The functionality of *egrep* and *fgrep*, (see *egrep*) may eventually be provided in *grep*, and those two commands discontinued.

SEE ALSO

egrep, *sed*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

iconv – codeset conversion

SYNOPSIS

iconv -f fromcode -t tocode [file]

DESCRIPTION

The *iconv* utility converts the encoding of characters in *file* from one codeset to another and writes the results to standard output. The input and output codesets are identified by *fromcode* and *toencode* respectively. If no *file* argument is specified on the command line, *iconv* reads the standard input.

Character encodings in either codeset may include single-byte values (e.g., for ISO standard **ISO 8859-1:1987** characters) or multi-byte values (e.g., for certain characters in ISO standard **ISO 6937:1983**). The results of specifying invalid characters in the input stream are implementation defined.

Settings of *fromcode* and *toencode* are implementation defined.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXAMPLE

The following example converts the contents of file **mail.x400** from codeset **ISO 6937:1983** to **ISO 8859-1:1987**, and stores the results in file **mail.local**.

```
iconv -f IS6937 -t IS8859 mail.x400 > mail.local
```

CHANGE HISTORY

First released in Issue 3.

NAME

`id` – print user and group IDs and names

SYNOPSIS

OF **id**

DESCRIPTION

The *id* utility writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are written.

SEE ALSO

logname, **Volume 2, XSI System Interfaces and Headers**, *getuid()*, *getgid()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

join – join two files on identical-valued fields

SYNOPSIS

join [options] file1 file2

DESCRIPTION

The *join* utility performs an “equality join” on the files *file1* and *file2*. If *file1* is –, the standard input is used in its place.

A field must be specified for each file as the “join field” on which the files are compared. There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*. This format can be changed by using the *-o* option (see below).

The files *file1* and *file2* must be sorted in the collating sequence of *sort*, see *sort*, on the fields on which they are to be joined, normally the first in each line.

The default input field separators are blank, tab, or newline. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the options below use the argument *n*. This argument should be a “1” or a “2” referring to either *file1* or *file2*, respectively. The following options are recognised:

- an* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- e s* Replace empty output fields by string *s*.
- jn m*
Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.
- o list*
Each output line comprises the fields specified in *list*, each element of which has the form *n.m* where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
- tc* Use character *c* as a separator, for both input and output. Every appearance of *c* in a line is significant.

APPLICATION USAGE

IN In an internationalised environment, the value of the *LC_COLLATE* environment variable must be equal to the value it had when the input files were sorted.

If *join* does not support selection of collating sequences via *LC_COLLATE*, the input files must be sorted according to the collating sequence of the “C” locale (see **Volume 3, XSI Supplementary Definitions, Chapter 7, C Program Locale**). Filenames that are numeric may cause conflict when the *-o* option is used immediately before listing filenames.

join

Utilities

SEE ALSO

awk, comm, sort, uniq.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an internationalised environment has been described.

NAME

kill – send a signal to a process

SYNOPSIS

kill [-signal] pid ...

DESCRIPTION

The *kill* utility sends the specified *signal* to the processes (or process groups) specified by *pid*. If process number 0 is specified, all processes in the process group are signalled. Process numbers can be found by using *ps*, see *ps*.

The argument *signal* must be specified as a numeric value; these values are implementation dependent. (See **APPLICATION USAGE** below.)

If no signal is specified, *kill* sends SIGTERM (terminate). This will terminate processes that do not catch or ignore the signal.

The specified process(es) must belong to the user unless the user has appropriate privileges.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Until symbolic naming is introduced, the following numeric values for the signals can be used:

- 1 SIGHUP
- 2 SIGINT
- 3 SIGQUIT
- 9 SIGKILL
- 15 SIGTERM

Refer to <signal.h> in **Volume 2, XSI System Interfaces and Headers** for the definitions of these signals.

FUTURE DIRECTIONS

The *kill* utility will be changed to use symbolic names rather than numeric values of signals. The old form will continue to be accepted for some time.

SEE ALSO

ps, **Volume 2, XSI System Interfaces and Headers**, *kill()*, <signal.h>.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

lex – generate programs for simple lexical analysis of text (DEVELOPMENT)

SYNOPSIS

MV UN lex [-ctvn] [file ...]

DESCRIPTION

The *lex* utility generates programs to be used in lexical processing of character input and may be used as an interface to *yacc*.

The input *file(s)*, which contain *lex* source code, contain a table of regular expressions, each with a corresponding action in the form of a C-language program fragment. Multiple input *files* are treated as a single file. When *lex* processes *file(s)*, this source is translated into a C-program. Normally *lex* writes the program it generates to the file *lex.yy.c*. If the *-t* option is used, the resulting program is written instead to the standard output. When the program generated by *lex* is compiled and executed, it will read character input from the standard input and partition it into strings that match the given expressions. When an expression is matched, the input string that was matched is left in an external character array, *yytext*, and the expression's corresponding program fragment, or action, is executed. The *lex* command also provides a count, *yyteng*, of the number of characters matched. During pattern matching the set of patterns will be searched for a match in the order in which they appeared in the *lex* source and the single longest possible match will be chosen at any point in time. Among rules that match the same number of characters, the rule given first will be matched.

The program generated by *lex*, e.g., *lex.yy.c*, should be compiled and loaded with the *lex* library (using the *-ll* option with *cc*).

MV UN The option *-c* indicates C-language actions and is the default; *-t* causes the program generated to be written instead to standard output; *-v* provides a one-line summary of statistics of the finite state machine generated, and *-n* will not print out the *-v* summary.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p <i>n</i>	number of positions is <i>n</i>
%n <i>n</i>	number of states is <i>n</i>
%e <i>n</i>	number of parse tree nodes is <i>n</i>
%a <i>n</i>	number of transitions is <i>n</i>
%k <i>n</i>	number of packed character classes is <i>n</i>
%o <i>n</i>	size of the output array is <i>n</i>

The use of one or more of the above automatically implies the *-v* option, unless the *-n* option is used.

The general format of *lex* source is:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

The definitions and the user subroutines may be omitted. The first %% is required to mark the beginning of the rules (regular expressions and actions); the second %% is required only if user subroutines follow.

In the definitions, any lines beginning with whitespace, and any lines included between lines containing only %{ and %}, are assumed to contain only C text and are copied unchanged into the external definition area of the *lex.yy.c* file. Such lines may also appear between the first %% delimiter and the first *lex* rule but, in this case, they are copied unchanged into the internal definition area of the program *yylex()* generated by *lex*. All text after the second %% delimiter is copied unchanged to *lex.yy.c*.

Definitions

Definitions must appear before the first %% delimiter. Any line in this section not contained between %{ and %} lines and beginning in column 1 is assumed to define a *lex* substitution string. The format of these lines is

```
name substitute
```

The *name* must begin with a letter and be followed by at least one blank or tab. The *substitute* will replace the string *{name}* when it is used in a rule. The braces do not imply parentheses; only string substitution is done.

Rules

The rules in *lex* source files are a table in which the left column contains regular expressions and the right column contains actions and program fragments to be executed when the expressions are recognised.

```
re whitespace action
re whitespace action
...
```

Because the regular expression, *re*, portion of a rule is terminated by the first blank or tab, any blank or tab used within a regular expression must be quoted (its special meaning escaped). That is, it must appear within double quotes or square brackets or must be preceded by a backslash character.

The program fragment which is the action associated with a particular *re* may extend across several lines if it is enclosed in braces:

```
re whitespace { program statement
                program statement }
```

Regular Expressions

The *lex* utility supports extended regular expression syntax, with some

additional expressions. Below is a summary of the additions.

1. Characters surrounded by double quotes, "...", have no special meaning.
2. The notation *<s>r* matches the regular expression *r* only when the program is in the start condition (state), *s*.
3. The notation *r/x* matches the regular expression *r* only if it is followed by the regular expression *x*. (Note this is *r* in the context of *x* and only *r* is matched.)
4. The notation */S* matches the substitution of *S* from the *definitions* section.

Actions

The default action when a string in the input to a *lex.yy.c* program is not matched by any expression is to copy the string to the output. Because the default behaviour of a program generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that has just `%%` will generate a C-language program that simply copies the input to the output unchanged. A null C-language statement, the statement `;;`, may be specified as an action in a rule. Any string in the *lex.yy.c* input that matches the pattern portion of such a rule will be effectively ignored or skipped.

Three special actions are available; `|`, `REJECT` and `ECHO`. The action `|` means that the action for the next rule is the action for this rule. `ECHO` prints the string *yytext* on the output. Normally only a single expression is matched by a given string in the input. `REJECT` means "continue to the next expression that matches the current input" and causes whatever rule was second choice after the current rule to be executed for the same input. Thus, it allows multiple rules to be matched and executed for one input string or overlapping input strings. For example, given the expressions *xyz* and *yz* and the input *xyz*, normally only one pattern, *xyz*, would match and the next attempted match would start at *z*. If the last action in the *xyz* rule is `REJECT`, both this rule and the *yz* rule would be executed.

The *lex* command provides several routines that can be used in the *lex* source program: *yymore()*, *yylless(n)*, *input()*, *output(c)* and *unput(c)*

The function *yymore()* may be called to indicate that the next input string recognised is to be concatenated onto the end of the current string in *yytext* rather than overwriting it in *yytext*. The function *yylless(n)* returns to the input some of the characters matched by the currently successful expression. The argument *n* indicates the number of initial characters in *yytext* to be retained; the remaining trailing characters in *yytext* are returned to the input.

input()

return the next character from the input. The function *input* returns 0 on end of file.

unput(c)

push the character *c* back onto the input stream to be read later by *input()*.

output(c)

write the character *c* on the output.

To perform custom processing when the end of input is reached, users may supply their own *yywrap()* function. The function *yywrap()* is called whenever *lex.yy.c* reaches an end-of-file. If *yywrap()* returns a one, *lex.yy.c* continues with the normal wrap-up on end of input. The default *yywrap()* always returns a one. If the user wants *lex.yy.c* to continue processing with another source of input, then a *yywrap()* must be supplied that arranges for the new input and returns a zero. These routines may be redefined by the user.

The external names generated by *lex* all begin with the prefix *yy* or *YY*.

The program generated by *lex* is named *yylex()*; if the user does not supply a main routine, the default *main()* routine calls *yylex()*. If the user supplies a *main()* routine, it should call *yylex()*.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXAMPLE

```
D      [0-9]
%%
if      printf("IF statement\n");
[a-z]+ printf("tag, value %s\n",yytext);
0{D}+  printf("octal number %s\n",yytext);
{D}+   printf("decimal number %s\n",yytext);
"++"   printf("unary op\n");
"+"    printf("binary op\n");
"/*"   {
loop:
while (input() != '*');
switch (input()) {
case '/':
break;
case '*':
unput('*');
default:
goto loop;
}
}
```

SEE ALSO

cc, *yacc*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

The *line* utility copies one line (up to and including a newline) from the standard input and writes it on the standard output. It always writes at least a newline.

EXIT STATUS

Exit status is:

- 0 normal exit
- 1 EOF on input

APPLICATION USAGE

The *line* utility is often used within command scripts to read from the user's terminal.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

lint – a C-language program checker (**DEVELOPMENT**)

SYNOPSIS

lint [options] file ...

DESCRIPTION

The *lint* utility attempts to detect features of the C-language program *file* that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

The options are described below.

Arguments whose names end with *.c* are taken to be C-language source files.

Arguments whose names end with *.ln* are taken to be the result of an earlier invocation of *lint* with either the *-c* or the *-o* option used. The *.ln* files are analogous to *.o* (object) files that are produced by the *cc* command when given a *.c* file as input.

Files with other suffixes are warned about and ignored.

The *lint* utility will take all the *.c*, *.ln* and *llib-lxxx* (specified by *-lxxx*) files and process them in their command line order. By default, *lint* appends the standard C-language lint library to the end of the list of files. However, if the *-p* option is used, the portable C-language lint library (*llib-port.ln*) is appended instead. When the *-c* option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the *-c* option is used, the *.ln* files and the lint libraries are ignored.

Any number of *lint* options may be used, in any order, intermixed with filename arguments. The following options are used to suppress certain kinds of complaints:

- | | | |
|----|-----------|---|
| PI | -a | Suppress complaints about assignments of long values to variables that are not long. |
| PI | -b | Suppress complaints about <i>break</i> statements that cannot be reached. (Programs produced by <i>lex</i> or <i>yacc</i> will often result in many such complaints.) |
| PI | -h | Do not apply heuristic tests that attempt to intuit bugs, improve style and reduce waste. |
| PI | -u | Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running <i>lint</i> on a subset of files of a larger program.) |
| PI | -v | Suppress complaints about unused arguments in functions. |
| PI | -x | Do not report variables referred to by external declarations but never used. |

The following arguments alter *lint*'s behaviour:

- lxxx** Include additional lint library *xxx* (e.g., *-lm* for the maths library).
- n** Do not check compatibility against either the standard or the portable *lint* library.
- p** Attempt to check portability.
- c** Cause *lint* to produce a *.ln* file for every *.c* file on the command line. These *.ln* files are the product of *lint*'s first pass only and are not checked for inter-function compatibility.
- o lib** Cause *lint* to create a lint library with the name *lib*. The *-c* option nullifies any use of the *-o* option. The lint library produced is the input that is given to *lint*'s second pass. The *-o* option simply causes this file to be saved in the named lint library. To produce the lint library without extraneous messages, use of the *-x* option is suggested. The *-v* option is useful if the source file(s) for the lint library are just external interfaces. These option settings are also available through the use of "lint comments" (see below).

The *-D*, *-U* and *-I* options of the C-language preprocessor phase of *cc* are also recognised as separate arguments.

The *-g* and *-O* options of *cc* are also recognised as separate arguments. These options are ignored but, by recognising these options, *lint*'s behaviour is closer to that of the *cc* command. Other options are warned about and ignored.

The preprocessor symbol *lint* is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol *lint* should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C-language source will change the behaviour of *lint*:

/*NOTREACHED*/

at appropriate points stop comments about unreachable code. (This comment is typically placed just after calls to functions like *exit()*.)

/*VARARGS*n/**

suppress the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/

turn on the *-v* option for the next function.

/*LINTLIBRARY*/

at the beginning of a file shut off complaints about unused functions and function arguments in this file. This is equivalent to using the *-v* and *-x* options.

The *lint* utility produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the *-c* option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source filename will be printed followed by a question mark.

The behaviour of the *-c* and the *-o* options allows for incremental use of *lint* on a set of C-language source files. Generally, *lint* is invoked once for each source file with the *-c* option. Each of these invocations produces a *.In* file which corresponds to the *.c* file and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the *-c* option), listing all the *.In* files with the needed *-lxxx* options. This will print all the inter-file inconsistencies.

This scheme works well with *make*; it allows *make* to be used to run through *lint*, only the source files that have been modified since the last time the set of source files were checked by *lint*.

APPLICATION USAGE

On some implementations, the behaviour of the *-a*, *-b*, *-h*, *-u*, *-v* and *-x* options is reversed, in that their presence enables the messages.

SEE ALSO

cc, *make*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

logname

Utilities

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

The *logname* utility writes the user's login name on the standard output.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

lp, cancel – send/cancel line printer requests

SYNOPSIS

PI UN **lp** [**-c**] [**-d dest**] [**-m**] [**-n number**] [**-o option**] [**-s**] [**-t title**] [**-w**] [**file ...**]

UN **cancel** [**id ...**] [**printer ...**]

DESCRIPTION

lp

The *lp* utility arranges for the named *file* or files and associated information (collectively called a *request*) to be printed by a line printer. If no filenames are mentioned, the standard input is assumed. The filename *-* stands for the standard input and may be supplied on the command line in conjunction with named files. The order in which files appear is the same order in which they will be printed.

OF The *lp* utility associates a unique *id* with each request and writes it on the standard output. This *id* can be used later to cancel, see *cancel*, or find the status, see *lpstat*, of the request.

The following options to *lp* may appear in any order and may be intermixed with filenames:

-c Make copies of the files to be printed immediately when *lp* is invoked. Normally, files will not be copied, but will be linked whenever possible. If the *-c* option is not given, then the user should be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the *-c* option, any changes made to the named files after the request is made but before it is printed will be reflected in the printed output. On some systems, *-c* may be on by default.

-d dest

Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, the request will be printed only on that specific printer. If *dest* is a class of printers, the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted, see *lpstat*. By default, *dest* is taken from the environment variable *LPDEST* (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems, see *lpstat*.

UN **-m** Send mail, see *mail*, after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n number

Print *number* copies (default of 1) of the output.

UN **-o option**

Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the *-o* option more than once.

- PI -s Suppress messages from *lp* such as "request id is ...".
- UN -t*title*
 Print *title* on the banner page of the output.
- UN -w Write a message on the user's terminal after the files have been printed. If the user is not logged in, then mail will be sent instead.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

cancel

- UN The *cancel* utility cancels line printer requests that were made by the *lp* command. The command line arguments may be either request *ids* (as returned by *lp*) or printer names (for a complete list, use *lpstat*). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a printer cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

FUTURE DIRECTIONS

The date and time strings printed on the banner page (if any) will be output in an implementation-defined manner. In the future, the format may become dependent on the setting of the *LC_TIME* environment variable.

SEE ALSO

lpstat, *mail*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

lpstat – print line printer status information

SYNOPSIS

UN OF
PI **lpstat** [**options**] [**id ...**]

DESCRIPTION

The *lpstat* utility prints information about the current status of the line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp* by the user. Any arguments that are not *options* are assumed to be request *ids* as returned by *lp*, see *lp*. The *lpstat* utility prints the status of such requests. The *options* may appear in any order and may be repeated and intermixed with other arguments. Some of the options below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
lpstat -u"user1, user2, user3"
```

The omission of a *list* following such options causes all information relevant to the option to be printed, for example:

```
lpstat
```

prints the status of all output requests.

-a [*list*]

Print acceptance status of destinations for output requests. The *list* argument is a list of intermixed printer names and class names.

-c [*list*]

Print class names and their members. The *list* argument is a list of class names.

-d Print the system default destination for output requests.

-o [*list*]

Print the status of output requests. The *list* argument is a list of intermixed printer names, class names, and request *ids*.

-p [*list*]

Print the status of printers. The *list* argument is a list of printer names.

-r Print the status of the line printer request scheduler.

-s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

-t Print all status information.

-u [*list*]

Print status of output requests for users. The *list* argument is a list of login names.

-v[*list*]

Print the names of printers and the pathnames of the devices associated with them. The *list* argument is a list of printer names.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN *LC_TIME* determines the format of date and time strings output when displaying line printer status information with the *-a*, *-o*, *-p*, *-t* or *-u* options, e.g.,

`lpstat -t`

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

SEE ALSO

lp.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

ls – list contents of directory

SYNOPSIS

ls [options] [file ...]

DESCRIPTION

For each *file* argument, if it is a directory, *ls* lists the contents of the directory; if it is a file, *ls* repeats its name and gives any other information requested. The output is sorted in ascending collation order by default (see **Environment Variables**). When no *files* are specified, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but files appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line; the options *-C* and *-x* enable multi-column formats, and the *-m* option enables stream output format in which files are listed across the page, separated by commas.

In order to determine output formats for the *-C*, *-x* and *-m* options, *ls* uses the environment variable *COLUMNS* to determine the number of character positions available on one output line. If this variable is not set, *ls* may optionally attempt to determine this information from a system-dependent database, keyed on the value of the *TERM* environment variable. If this information cannot be obtained, 80 columns is assumed.

There are a large number of options:

- OF *-C* Multi-column output with entries sorted down the columns.
- OF *-F* Put a slash, /, after each filename if that file is a directory and put an asterisk, *, after each filename if that file is executable. For other file types, other symbols may be printed.
- OF *-R* Recursively list subdirectories encountered.
- a* List all entries; usually entries whose names begin with a period, ".", are not listed.
- c* Use time of last modification of the file status information (file created, mode changed, etc.) for sorting (*-t*) or printing (*-l*).
- d* If an argument is a directory, list only its name (not its contents); often used with *-l* to get the status of a directory.
- f* Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off *-l*, *-t*, *-s* and *-r*, and turns on *-a*; the order is the order in which entries appear in the directory.
- OF *-g* The same as *-l*, except that the owner is not printed.
- i* For each *file*, print its file serial number in the first column of the report.

- OF **-l** List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each *file* (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
- OF **-m** Stream output format; files are listed across the page, separated by commas.
- OF UN **-n** The same as **-l**, except that the owner's *UID* and group's *GID* numbers are printed, rather than the associated character strings.
- OF **-o** The same as **-l**, except that the group is not printed.
- p** Put a slash, /, after each filename if that file is a directory.
- q** Force non-printing characters (in filenames) to be displayed as the character "?".
- r** Reverse the order of sort to get descending collation order, or oldest first, as appropriate.
- OF **-s** Give total number of 512-byte units consumed by each *file*.
- t** Sort by time modified (latest first) instead of by name.
- u** Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).
- OF UN **-x** Multi-column output with entries sorted across rather than down the page.

The mode printed under the **-l** option consists of 10 characters that are interpreted as described below.

The first character is:

- d** if the entry is a directory
- b** if the entry is a block special file
- c** if the entry is a character special file
- p** if the entry is a FIFO (named pipe) special file
- if the entry is an ordinary file

On some implementations, other characters may be printed for implementation dependent file types.

The next 9 characters are presented as three sets of three characters each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file, and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively.

The permissions are indicated as follows:

- r** if the file is readable
- w** if the file is writable
- x** if the file is executable (also see below)
- if the indicated permission is not granted

The group-execute permission character is given as `s` if the file has set-group-ID mode; likewise, the user-execute permission character is given as `s` if the file has set-user-ID mode. These are given as `S` (capitalised) if the corresponding execute permission is not set.

On some implementations, other characters may be printed to indicate implementation dependent permissions.

When the sizes of the files in a directory are listed, a total count of 512-byte units consumed by the directory is printed.

This utility operates in an 8 bit transparent manner. See **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN `LC_COLLATE` determines the order in which the output is sorted, `LC_CTYPE` determines which characters are classified as non-printing for the `-q` option, and `LC_TIME` determines the format and contents of date and time strings output by the `-g`, `-l`, `-n` and `-o` options.

If `LC_COLLATE`, `LC_CTYPE` or `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default for each unset/empty variable. If `LANG` is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

Currently not all systems report in terms of 512-byte units. This situation may change in the future.

SEE ALSO

chmod, *find*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

m4 – macro processor (DEVELOPMENT)

SYNOPSIS

m4 [options] [file ...]

DESCRIPTION

The *m4* utility is a macro processor intended as a front end for the C-language and other languages. Each of the argument files is processed in order; if there are no files, or if a filename is *-*, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

- s Enable line sync output for the *cc* preprocessor phase (i.e., *#line* directives). This option must appear before any filenames and before the following options.
- D*name*[=*val*] Define *name* to *val* or to null in *val*'s absence.
- U*name* Undefine *name*.

Macro calls have the form:

name(arg1, arg2, ..., arg*n*)

The “(” must immediately follow the name of the macro. If the name of a defined macro is not followed by a “(”, it is deemed to be a call of that macro with no arguments. Potential macro names consist of letters, digits and underscore *_*, where the first character is not a digit.

Leading unquoted blanks, tabs and newlines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognised, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null.

Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses that appear within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

The *m4* utility makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define

the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of `$n` in the replacement text, where *n* is a digit, is replaced by the *n*th argument. Argument 0 is the name of the macro, missing arguments are replaced by the empty string, `$#` is replaced by the number of arguments, `$*` is replaced by a list of all the arguments separated by commas, and `$@` is like `$*`, but each argument is quoted (with the current quotes).

undefine

removes the definition of the macro named in its argument.

defn returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef

like *define*, but saves any previous definition.

popdef

removes current definition of its argument(s), exposing the previous one, if any.

ifdef if the first argument is defined, the value is the second argument, otherwise it is the third. If there is no third argument, the value is null.

shift returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote

changes quote symbols to the first and second arguments. The symbols may be up to five characters long. The command *changequote* without arguments restores the original values (i.e., ``` and `~`).

changecom

changes left and right comment markers from the default `#` and newline. With no argument, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert

the *m4* utility maintains ten output streams, numbered 0–9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 to 9 is discarded.

undivert

causes immediate output of text from diversions named as arguments, or all diversions if no arguments are present. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum

returns the value of the current output stream.

dnl reads and discards characters up to and including the next newline.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponent); bitwise &, |, ^ and ~; relationals and parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr

returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit

transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include

returns the contents of the file named in the argument.

sinclude

is identical to *include*, except that it says nothing if the file is inaccessible.

syscmd

executes the system command given in the first argument. No value is returned.

sysval

is the return code from the last call to *syscmd*.

maketemp

fills in a string of XXXXX in its argument with the current process ID.

m4exit

causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap

argument 1 will be pushed back at final EOF; example: *m4wrap(`cleanup()`)*

errprint

prints its argument on the diagnostic output file.

dumpdef

prints current names and definitions for the named items, or for all if no arguments are given.

traceon

with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff

turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

mail – send or read mail

SYNOPSIS

UN **mail** [-epqr] [-f file]
mail [-t] name ...

DESCRIPTION

Reading Mail

The *mail* utility without arguments prints a user's mail, message-by-message. Mail is stored in the user's individual mailfile. For each message, the user is given a prompt and a line is read from the standard input to determine the disposition of the message:

<newline>

Go on to next message.

+ Same as **<newline>**.

d Delete message and go on to next message.

p Display message again.

- Go back to previous message.

s [file]

Save message in the named *file* (**mbox** is default).

PI **w** [file]

Save message (on some implementations, without its header) in the named *file* (**mbox** is default).

PI **m** [name ...]

Mail the message to the named users; the default is the user.

PI **q**

Store undeleted mail and stop.

<EOF>

Same as *q*.

x Put all mail back in the mailfile unchanged and stop.

!command

Escape to the command interpreter to execute *command*.

UN *****

Display a command summary.

The optional arguments alter the printing of the mail:

-e Causes mail not to be printed. An exit value is returned; see **EXIT STATUS**.

UN **-p**

Causes all mail to be printed without prompting for disposition.

UN **-q**

Causes *mail* to terminate after interrupts. Normally an interrupt causes termination only of the message being printed.

UN **-r**

Causes messages to be printed in first-in, first-out order.

`-f file`

Causes *mail* to use *file* (e.g., **mbox**) instead of the default mailfile.

Sending Mail

When *names* (user login names) are given, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a `"."`) and adds it to each user's mailfile. The message is preceded by the sender's name and a postmark. Lines in the message that begin with the sequence `"From "` are preceded with a `>`.

UN The `-t` option causes the message to be preceded by all users the *mail* is sent to.

If a user being sent mail is not recognised, or if *mail* is interrupted during input, the message is saved in the file **dead.letter** to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time it is needed, erasing the previous contents of **dead.letter**.

PI It may also be possible to send mail to remote systems using system-specific naming conventions.

PI There are implementation-specific mechanisms that can be used to cause all mail sent to the user to be forwarded to one or more other destinations.

Environment Variables

IN *LC_TIME* determines the format and contents of date and time strings displayed by the *mail* utility.

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

If *mail* is invoked with the `-e` option, the following exit values are returned:

0	the user has mail
1	the user has no mail

APPLICATION USAGE

The forwarding facility is especially useful to forward all of a person's mail to one machine in a multiple-machine environment.

Delivery of messages to remote systems requires the existence of communications paths to such systems. These may not exist.

The location of stored mail on exiting from *mail* using the *q* command differs between implementations and may be either the user's mailfile or the user's **mbox**.

In the description of reading mail, the phrase "go on to next message" may or may not imply the displaying of the next message.

IN **ISO 8859-1:1987** is defined as the transmission codeset for communication between X/Open systems. However, the 8-bit operation of the *mail* utility cannot be guaranteed in all circumstances. For example, inter-system mail may be restricted to

7-bit data by the underlying network, 8-bit data may not be portable to non-internationalised systems, etc. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an internationalised environment has been described.

NAME

mailx – interactive message processing system (OPTIONAL)

SYNOPSIS

UN **mailx** [*options*] [*name* ...]

DESCRIPTION

The *mailx* utility provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system mailbox for that user. When *mailx* is called to read messages, the mailbox is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's *HOME* directory (see *MBOX* in *Environment Variables* below for a description of this file). Messages remain in this file until specifically removed.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the mailbox. Command-line options are:

–e Test for presence of mail. The command *mailx* prints nothing and exits with a successful return code if there is mail to read.

–f [*filename*]
Read messages from *filename* instead of mailbox. If no *filename* is specified, the mailbox is used.

–F Record the message in a file named after the first recipient. Overrides the *record* variable, if set (see *Environment Variables*).

PI UN –h*number*
The number of network “hops” made so far. This is provided for network software to avoid infinite delivery loops.

–H Print header summary only.

–i Ignore receipt of SIGINT. See also *ignore* (*Environment Variables*).

–n Do not initialise from the system default *mailx.rc* file.

–N Do not print initial header summary.

PI UN –r *address*
Pass *address* to network delivery software. All tilde commands are disabled.

–s *subject*
Set the Subject header field to *subject*.

–u *user*
Read *user*'s mailbox. This is only effective if *user*'s mailbox is not read protected.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see **Commands** below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the `~` escape character followed by a single command letter and optional arguments. See **Tilde Escapes** for a summary of these commands.

At any time, the behaviour of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the *set* and *unset* commands. See **Environment Variables** below for a summary of these parameters.

Regular commands are of the form:

[command] [msglist] [arguments]

If no command is specified in *command mode*, *print* is assumed. In *input mode*, commands are recognised by the escape character and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number and there is at any one time the notion of a "current" message, marked by `>` in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

- n* Message number *n*.
- .* The current message.
- ^* The first undeleted message.
- \$* The last message.
- ** All messages.
- n-m* An inclusive range of message numbers.
- user* All messages from *user*.
- /string*
 All messages with *string* in the subject line (case ignored).
- :c* All messages of type *c*, where *c* is one of:
 - d* deleted messages
 - n* new messages
 - o* old messages
 - r* read messages
 - u* unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, can be specified with metacharacters understood by the command interpreter. Special characters are recognised by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file to initialise certain parameters, then from a private start-up file (*\$HOME/.mailrc*) for personalised variables. Most regular commands are valid inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: *!*, *Copy*, *edit*, *followup*, *Followup*, *hold*, *mail*, *preserve*, *reply*, *Reply*, *shell* and *visual*. Any errors in the start-up file cause the remaining lines in the file to be ignored.

Commands

The following is a complete list of *mailx* commands, most of which can be abbreviated. For each command, the abbreviated form is shown underneath the full form.

!shell-command

Escape to the command interpreter. See *SHELL* (Environment Variables).

comment

Null command (comment). This may be useful in *.mailrc* files.

= Print the current message number.

? Print a summary of commands.

alias alias name...

a alias name ...

Declare an alias for the given *names*. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates [name ...]

alt [name ...]

Declare a list of alternate names for the user's login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, *alternates* prints the current list of alternate names. See also *allnet* (Environment Variables).

cd [directory]

chdir [directory]

ch [directory]

Change directory. If *directory* is not specified, *\$HOME* is used.

copy [*filename*]

c [*filename*]

copy [*msglist*] *filename*

c [*msglist*] *filename*

Copy messages to the file *filename* without marking the messages as saved. Otherwise equivalent to the *save* command.

Copy [*msglist*]

C [*msglist*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the *Save* command.

delete [*msglist*]

d [*msglist*]

Delete messages from the mailbox. If *autoprint* is set, the next message after the last one deleted is printed (see **Environment Variables**).

discard [*header-field ...*]

di [*header-field ...*]

Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are *status* and *cc*. The fields are included when the message is saved. The *Print* and *Type* commands override this command.

dp [*msglist*]

dt [*msglist*]

Delete the specified messages from the mailbox and print the next message after the last one deleted. Roughly equivalent to a *delete* command followed by a *print* command.

echo *string ...*

ec *string ...*

Echo the given strings (like *echo*).

edit [*msglist*]

e [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the *EDITOR* variable is used to get the name of the editor (see **Environment Variables**). Default editor is *ed*.

exit

ex Exit from *mailx*, without changing the mailbox. No messages are saved in the *mbox* (see also *quit*).

file [*filename*]

fi [*filename*]

folder [*filename*]

fold [*filename*]

Quit from the current file of messages and read in the file *filename*. Several special characters are recognised when used as *filenames*, with the following substitutions:

%	the current mailbox.
%user	the mailbox for user.
#	the previous file.
&	the current mbox .

Default file is the current mailbox.

folders

Print the names of the files in the directory set by the *folder* variable (see **Environment Variables**).

followup [*message*]

fo [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the *record* variable, if set. See also the Followup, Save and Copy commands and *outfolder* (**Environment Variables**).

Followup [*msglist*]

F [*msglist*]

Responds to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the *followup*, Save and Copy commands and *outfolder* (**Environment Variables**).

from [*msglist*]

f [*msglist*]

Print the header summary for the specified messages.

group *alias name ...*

g *alias name ...*

Declare an alias for the given *names*. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

headers [*message*]

h [*message*]

Print the page of headers which includes the message specified. The *screen* variable sets the number of headers per page (see **Environment Variables**). See also the **z** command.

help Print a summary of commands.

```
hold [msglist]
ho [msglist]
preserve [msglist]
pre [msglist]
```

Hold the specified messages in the mailbox.

```
if s | r
mail-command ...
else mail-command ...
endif
```

```
i s | r
mail-command ...
el mail-command ...
en
```

Conditional execution, where *s* will execute following *mail-commands*, up to an *else* or *endif*, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

```
ignore [header-field ...]
ig [header-field ...]
```

Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are *status* and *cc*. All fields are included when the message is saved. The *Print* and *Type* commands override this command.

```
list
```

```
l      Print all commands available. No explanation is given.
```

```
mail name ...
m name ...
```

Mail a message to the specified users.

```
mbox [msglist]
mb [msglist]
```

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See *MBOX (Environment Variables)* for a description of this file. See also the *exit* and *quit* commands.

```
next [message]
n [message]
```

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]

pi [*msglist*] [*shell-command*]

| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the *cmd* variable. If the *page* variable is set, a form-feed character is inserted after each message (see *Environment Variables*).

preserve [*msglist*]

pre [*msglist*]

hold [*msglist*]

ho [*msglist*]

Preserve the specified messages in the mailbox.

Print [*msglist*]

P [*msglist*]

Print the specified messages, including all header fields, on the screen. Overrides suppression of fields by the *ignore* command.

print [*msglist*]

p [*msglist*]

Print the specified messages. If *crt* is set, the messages longer than the number of lines specified by the *crt* variable are paged through the command specified by the *PAGER* environment variable. The default command is *pg*. (See *Environment Variables*.)

quit

q

Exit from *mailx*, storing those messages that were read in the *mbox*, and those that were unread in the mailbox. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]

Respond [*msglist*]

R [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If *record* is set to a filename, the response is saved at the end of that file (see *Environment Variables*).

reply [*message*]

r [*message*]

Reply to the specified *message*, including all other recipients of the message. If *record* is set to a filename, the response is saved at the end of that file (see *Environment Variables*).

Save [*msglist*]

S [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the *Copy*, *followup* and *Followup* commands and *outfolder* (**Environment Variables**).

save [*filename*]

s [*filename*]

save [*msglist*] *filename*

s [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the mailbox when *mailx* terminates unless *keepsave* is set (see **Environment Variables** and the *exit* and *quit* commands).

set [*name*]

se [*name*]

set [*name=string*]

se [*name=string*]

set [*name=number*]

se [*name=number*]

Define a variable called *name*. The variable may be given a null, string or numeric value. With no arguments, *set* by itself prints all defined variables and their values. See **Environment Variables** for detailed descriptions of the *mailx* variables.

shell

sh Invoke an interactive command interpreter (see also *SHELL* (**Environment Variables**)).

size [*msglist*]

si [*msglist*]

Print the size in characters of the specified messages.

source *filename*

so *filename*

Read commands from the given file and return to command mode.

top [*msglist*]

to [*msglist*]

Print the top few lines of the specified messages. If the *toplines* variable is set, it is taken as the number of lines to print (see **Environment Variables**). The default is 5.

touch [*msglist*]

tou [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the **mbox** upon normal termination. See *exit* and *quit*.

Type [*msglist*]

T [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the *ignore* command.

type [*msglist*]

t [*msglist*]

Print the specified messages. If *crt* is set, the messages longer than the number of lines specified by the *crt* variable are paged through the command specified by the *PAGER* variable. The default command is *pg*. (See **Environment Variables**.)

undelete [*msglist*]

u [*msglist*]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If *autoprint* is set, the last message of those restored is printed (see **Environment Variables**).

unset *name* ...

uns *name* ...

Cause the specified variables to be erased. If the variable was imported from the execution environment (i.e., an environment variable), then it cannot be erased.

MV

version

ve Print the current version and release date.

visual [*msglist*]

v [*msglist*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the *VISUAL* variable is used to get the name of the editor (see **Environment Variables**).

write [*msglist*] *filename*

w [*msglist*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the *save* command.

xit

x Exit from *mailx*, without changing the mailbox. No messages are saved in the **mbox** (see also *quit*).

z[+|-]

Scroll the header display forwards or backwards one screen-full. The number of headers displayed is set by the *screen* variable (see **Environment Variables**).

Tilde Escapes

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character `~`. See *escape* (**Environment Variables**) for changing this special character.

- `~! shell-command`
Escape to the command interpreter.
- `~.` Simulate end of file (terminate message input).
- `~: mail-command`
- `~_mail-command`
Perform the command-level request. Valid only when sending a message while reading mail.
- `~?` Print a summary of tilde escapes.
- `~A` Insert the autograph string *Sign* into the message (see **Environment Variables**).
- `~a` Insert the autograph string *sign* into the message (see **Environment Variables**).
- `~b name ...`
Add the *name* to the blind carbon-copy (Bcc) list.
- `~c name ...`
Add the *name* to the carbon-copy (Cc) list.
- `~d` Read in the **dead.letter** file. See *DEAD* (**Environment Variables**) for a description of this file.
- `~e` Invoke the editor on the partial message. See also *EDITOR* (**Environment Variables**).
- `~f [msglist]`
Forward the specified messages. The messages are inserted into the message, without alteration.
- `~h` Prompt for Subject line and To, Cc and Bcc lists. If the field is displayed with an initial value, it may be edited as if it had just been typed.
- `~i string`
Insert the value of the named variable into the text of the message. For example, `~i Sign.` ,
- `~m [msglist]`
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.
- `~p` Print the message being entered.
- `~q` Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in **dead.letter**. See *DEAD* (**Environment Variables**) for a description of this file.
- `~r filename`
- `~<filename`
- `~<!shell-command`
Read in the specified file. If the argument begins with `!`, the rest of the string is taken as an arbitrary system command and is executed, with the standard output inserted into the message.

- ~s** *string* ...
Set the subject line to *string*.
- ~t** *name* ...
Add the given *name* to the To list.
- ~v** Invoke a preferred screen editor on the partial message. See also *VISUAL* (Environment Variables).
- ~w** *filename*
Write the partial message onto the given file, without the header.
- ~x** Exit as with **~q** except the message is not saved in **dead.letter**.
- ~|** *shell-command* Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the *shell-command* replaces the message.

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is *\$HOME/.mailrc*.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the *set* command at any time. The *unset* command may be used to erase variables.

allnet

All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is *noallnet*. See also the *alternates* command and the *metoo* variable.

append

Upon termination, append messages to the end of the **mbox** file instead of prepending them. Default is *noappend*.

askcc

Prompt for the Cc list after message is entered. Default is *noaskcc*.

asksub

Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint

Enable automatic printing of messages after *delete* and *undelete* commands. Default is *noautoprint*.

bang Enable the special-case treatment of "!"s in escape command lines as in *vi*. Default is *nobang*.

cmd=shell-command
Set the default command for the *pipe* command. No default value.

conv=conversion
Convert *uucp* addresses to the specified address style. Conversion is disabled by default. See also *sendmail* and the *-U* command line option.

crt=number
Pipe messages having more than *number* lines through the command specified by the value of the *PAGER* variable (*pg* by default). Disabled by default.

DEAD=filename
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is *\$HOME/dead.letter*.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is *nodebug*.

dot Take a "." on a line by itself during input from a terminal as end-of-file. Default is *nodot*.

EDITOR=shell-command
The command to run when the *edit* or *~e* command is used. Default is *ed*.

escape=c
Substitute *c* for the *~* escape character.

folder=directory
The directory for saving standard mail files. User-specified filenames beginning with a + are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash /, *\$HOME* is prepended to it. In order to use the + construct on a *mailx* command line, *folder* must be an exported environment variable. There is no default for the *folder* variable. See also *outfolder* below.

header
Enable printing of the header summary when entering *mailx*. Enabled by default.

hold Preserve all messages that are read in the mailbox instead of putting them in the standard *mbox* save file. Default is *nohold*.

ignore
Ignore receipt of SIGINT while entering messages. Handy for noisy dial-up lines. Default is *noignore*.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a "." on a line by itself or by the "~." command. Default is *noignoreeof*. See also *dot* above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the mailbox instead of deleting them. Default is *nokeepsave*.

MBOX=filename

The name of the file to save messages that have been read. The *xit* command overrides this function, as does saving the message explicitly in another file. Default is *\$HOME/mbox*.

metoo

If the user's login appears as a recipient, do not delete it from the list. Default is *nometoo*.

LISTER=shell-command

The command (and options) to use when listing the contents of the *folder* directory. The default is *ls*.

onehop

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder

Causes the files used to record outgoing messages to be located in the directory specified by the *folder* variable unless the pathname is absolute. Default is *nooutfolder*. See *folder* above and the *Save*, *Copy*, *followup* and *Followup* commands.

page Used with the *pipe* command to insert a form-feed after each message sent through the pipe. Default is *nopage*.

PAGER=shell-command

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*.

prompt=string

Set the *command mode* prompt to *string*. Default is "?".

quiet

Refrain from printing the opening message and version when entering *mailx*. Default is *noquiet*.

record=filename

Record all outgoing mail in *filename*. Disabled by default. See also *outfolder* above.

save Enable saving of messages in **dead.letter** on interrupt or delivery error. See *DEAD* for a description of this file. Enabled by default.

screen=number

Set the number of lines in a screenful of headers for the *headers* command.

sendmail=shell-command

Alternative command for delivering messages. Default is *mail*.

sendwait

Wait for background mailer to finish before returning. Default is *nosendwait*.

SHELL=shell-command

The name of a preferred command interpreter. Default is *sh*.

showto

When displaying the header summary, and the message is from the user, print the recipient's name instead of the author's name.

sign=string

The variable inserted into the text of a message when the *~a* (autograph) command is given. No default (see also *~i* (Tilde Escapes)).

Sign=string

The variable inserted into the text of a message when the *~A* command is given. No default (see also *~i* (Tilde Escapes)).

toplines=number

The number of lines of header to print with the *to* command. Default is 5.

VISUAL=shell-command

The name of a preferred screen editor. Default is system-dependent; usually *vi*.

IN Within bracketed filename patterns, e.g.,

~r [[:upper:]]*.mail

LC_COLLATE determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes. *LC_TIME* determines the format and contents of date and time strings displayed by the *mailx* utility.

If *LC_COLLATE*, *LC_CTYPE* or *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

If *mailx* is invoked with the *-e* option, the following exit values are returned:

0	the user has mail
1	the user has no mail

APPLICATION USAGE

IN ISO 8859-1:1987 is defined as the transmission codeset for communication between X/Open systems. However, the 8-bit operation of the *mailx* utility cannot be guaranteed in all circumstances. For example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data may not be portable to non-internationalised systems, etc. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines.

SEE ALSO

mail, pg, ls, vi.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an internationalised environment has been described.

The tilde escape *""* has been corrected to *"~|"*.

NAME

make – maintain, update, and regenerate groups of programs (DEVELOPMENT)

SYNOPSIS

UN **make** [**-f** *makefile*] [**-i**] [**-k**] [**-p**] [**-s**] [**-r**] [**-n**] [**-e**] [**-t**] [**-q**] [*name ...*]

DESCRIPTION

The options are interpreted as follows:

-f *makefile*

The argument *makefile* is assumed to be the name of a description file. A filename of **-** denotes the standard input.

UN **-p** Print out the complete set of macro definitions and target descriptions.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name *.IGNORE* appears in the description file.

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name *.SILENT* appears in the description file.

-r Do not use the built-in rules.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.

-e Environment variables override assignments within makefiles.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

-q Question. The *make* utility returns a zero or non-zero status code, depending on whether the target file is or is not up-to-date.

The following target names may be defined in the makefile, and are interpreted as follows:

.DEFAULT

If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name *.DEFAULT* are used if it exists.

.PRECIOUS

Dependants of this target will not be removed if SIGINT or SIGQUIT occur.

.SILENT

Same effect as the **-s** option.

.IGNORE

Same effect as the **-i** option.

The *make* utility executes commands in the makefile to update one or more target *names*. The argument *name* is typically a program, but may also be a macro definition, see **Environment**.

If no *-f* option is present, **makefile**, **Makefile** and the SCCS files **s.makefile** and **s.Makefile** are tried in order. If *makefile* is *-*, the standard input is used. More than one *-f makefile* argument pair may appear.

The utility *make* updates a target only if its dependants are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

The argument *makefile* contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a ":", then a (possibly null) list of prerequisite files or dependencies. Text following a ";" and all following lines that begin with a tab are commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. To continue commands across more than one line, all but the last line must be terminated by a backslash. Everything printed by *make* (except the initial tab) is passed directly to the command interpreter unchanged.

The symbols # and <newline> surround comments.

The following makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

Command lines are executed one at a time. The first one or two characters in a command can be the following: *-*, *@*, *-@* or *@-*. If *@* is present, printing of the command is suppressed. If *-* is present, *make* ignores an error. A line is printed when it is executed unless the *-s* option is present, or the entry *.SILENT:* is in the makefile, or unless the initial character sequence contains *@*. The *-n* option specifies printing without execution; however, if the command line has the string *\$(MAKE)* in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under **Environment**). The *-t* (touch) option updates the modified date of a file without executing any commands.

Each command line is executed in a separate invocation of the command interpreter.

Commands returning non-zero status normally terminate *make*. If the *-i* option is present, or the entry *.IGNORE:* appears in the makefile, or the initial character sequence of the command contains *-*, the error is ignored. If the *-k* option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

SIGINT and SIGQUIT cause the target to be deleted unless the target is a dependant of the special name *.PRECIOUS*.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and are processed as such. Macro definitions may also occur on the command line and in the makefile. Certain macros have internal defaults within *make*. The order of precedence depends on the setting of the *-e* option, as follows:

Without *-e*:

1. command line
2. makefile
3. environment
4. internal

With *-e*:

1. command line
2. environment
3. makefile
4. internal

The environment variable *MAKEFLAGS* is processed by *make* as containing any valid input option (except *-f* and *-p*) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, *MAKEFLAGS* always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the *-n* option is used, the command *\$(MAKE)* is executed anyway; hence, one can perform a *make -n* recursively on a whole software system to see what would have been executed. This is because the *-n* is put in *MAKEFLAGS* and passed to further invocations of *\$(MAKE)*.

Macros

Entries of the form *string1 = string2* are macro definitions. The macro *string2* is defined as all characters up to a comment character or an unescaped newline. Subsequent appearances of *\$(string1[:subst1=[subst2]])* are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional *:subst1=subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, newline characters and beginnings of lines. An example of the use of the substitute sequence is shown under **Libraries**.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

- \$*** The **\$*** macro stands for the filename part of the current dependant with the suffix deleted. It is evaluated only for inference rules.
- \$@** The **\$@** macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- \$<** The **\$<** macro is evaluated only for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target (i.e., the "manufactured", or dependant, filename). Thus, in the **.c.o** rule, the **\$<** macro would evaluate to the **.c** file. An example for making optimised **.o** files from **.c** files is:

```
.c.o:
    cc -c -O $*.c
or:
.c.o:
    cc -c -O $<
```

- \$?** The **\$?** macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules that must be rebuilt.
- %%** The **%%** macro is evaluated only when the target is an archive library member of the form **lib.a(file.o)**. In this case, **\$@** evaluates to **lib.a** and **%%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper-case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **\$(@D)** refers to the directory part of the string **\$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **\$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in the makefile, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. Inference rules in the makefile override the default rules.

UN

The internal rules used by *make* may be displayed by using the following commands:

```
make -fp - 2>/dev/null </dev/null
```


A `~` in the above rules refers to an SCCS file. Thus, the rule `.c~.o` would transform an SCCS C-language source file into an object file (`.o`). Because the `s.` of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the `~` is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (e.g., `.c:`) is the definition of how to build `x` from `x.c`. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., command procedures, simple C-language programs).

Additional suffixes are given as the dependency list for `.SUFFIXES`. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix-lists accumulate; `SUFFIXES:` with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
      cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the makefile.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, `CFLAGS`, `LFLAGS` and `YFLAGS` are used for compiler options to `cc`, `lex` and `yacc`, respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with `.c.o:` as the target and no dependants. Commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target that has no slashes in it and starts with `"."` is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib.a(file.o)** and **\$(LIB) (file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **\$(LIB)(file1.o file2.o)** is not valid. Rules pertaining to archive libraries have the form **.XX.a** where the **XX** is the suffix from which the archive member is to be made. The most common use of the archive interface follows. Here, we assume the source files are all C-type source:

```
lib.a:    lib.a(file1.o) lib.a(file2.o) lib.a(file3.o)
          @echo lib.a is now up-to-date

.c.a:
          $(CC) -c $(CFLAGS) $<
          ar rv $@ $*.o
          rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited, example of an archive library maintenance construction follows:

```
lib.a:    lib.a(file1.o) lib.a(file2.o) lib.a(file3.o)
          $(CC) -c $(CFLAGS) $(?:.o=.c)
          ar rv lib.a $?
          rm $?
          @echo lib.a is now up-to-date

.c.a;;
```

Here the substitution mode of the macro expansions is used. The **\$?** list is defined to be the set of object filenames (inside **lib.a**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. Note also the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C-language programs.

APPLICATION USAGE

The characters **=**, **:**, **"/**, **"**, and **@** in filenames may give trouble.

The appropriate transformations will be built into **make** on all X/Open systems for software developers using the C programming language, *lex*, *yacc*, shell programs and archives. The rules will also cater for the use of SCCS.

make

Utilities

SEE ALSO

cc, lex, sh, yacc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [y | n]

DESCRIPTION

The *mesg* utility with argument *n* prevents another user from writing to the invoking user's terminal (e.g., by using *write*). The command *mesg* with argument *y* reinstates write permission.

OF With no arguments, *mesg* reports the current state without changing it.

EXIT STATUS

Exit status is

- 0 messages are receivable
- 1 messages are not receivable
- 2 error

SEE ALSO

wall, *write*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

mkdir – make a directory

SYNOPSIS

mkdir *dirname*...

DESCRIPTION

The *mkdir* utility creates the specified directories. Standard entries, "." for the directory itself and ".." for its parent, are made automatically.

The *mkdir* utility requires write permission in the parent directory.

The *mkdir* utility performs actions equivalent of: for each *dirname* argument, the *mkdir()* function is called with the argument *path* set to *dirname* and *mode* set to 0777.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXIT STATUS

Exit status is

0	all directories were successfully made
non-0	otherwise

SEE ALSO *rm*, **Volume 2, XSI System Interfaces and Headers**, *mkdir()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

mkfifo – make FIFO special file

SYNOPSIS

mkfifo *path* ...

DESCRIPTION

The *mkfifo* utility creates the FIFO special files named by its argument list. The arguments are taken sequentially, in the order specified; and each FIFO special file is either created completely or, in the case of an error or signal, not created at all.

The *mkfifo* utility performs actions equivalent of: for each *path* argument, the *mkfifo()* function is called with the argument *path* set to *path* and *mode* set to 0666.

EXIT STATUS

The *mkfifo* utility exits with one of the following values:

- 0 All FIFO special files were created normally.
- >0 The utility was aborted due to errors in creating one or more FIFO special files.

APPLICATION USAGE

The *mkfifo* utility will terminate without action if it is passed a single invalid argument. In the case of an argument list containing more than one *path*, it may continue processing all the members in that list and display an error message at completion if one or more invalid arguments have been encountered.

SEE ALSO

umask, **Volume 2, XSI System Interfaces and Headers**, *mkfifo()*.

CHANGE HISTORY

First released in Issue 3.

NAME

newgrp – change to a new group (OPTIONAL)

SYNOPSIS

newgrp [-] [group]

DESCRIPTION

The *newgrp* command changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs.

Only exported environment variables retain their values after invoking *newgrp*. Otherwise, variables with a default value will be reset to that default, see *sh*.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

The following statements are true if the command interpreter named in the specified user's password file entry is *sh*. If the first argument to *newgrp* is *-*, the environment will be changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in the Group Database as being a member of that group.

The *newgrp* utility will fail if the user is not listed in the Group Database as being a member of that group and the group does not have a password.

APPLICATION USAGE

There is no convenient way to enter a password into the Group Database. Use of group passwords is not encouraged, because by their very nature they encourage poor security practices. Group passwords may disappear in the future.

A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with *newgrp*, which in turn overlays itself with a new shell after changing group. On some systems, however, this may not occur and *newgrp* may be invoked as a subprocess.

The *newgrp* command is intended only for use from an interactive terminal. It does not offer a useful interface for the support of applications.

SEE ALSO

sh, Volume 2, XSI System Interfaces and Headers, *exec*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

nl – line numbering filter

SYNOPSIS

nl [file]

UN nl [-h`type`] [-b`type`] [-f`type`] [-v`startnum`] [-i`incr`] [-p] [-l`num`] [-s`sep`]
[-w`width`] [-n`format`] [-d`delim`] [file]

DESCRIPTION

The *nl* utility reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left. Additional functionality may be provided in accordance with the command options in effect.

The *nl* utility views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The starts of logical page sections are signalled by input lines containing nothing but the following delimiter character(s):

Line	Start of
\:\:\:	header
\:\:	body
\:	footer

Unless otherwise specified, *nl* assumes the text being read is in a single logical page body.

Options may appear in any order and may be intermingled with an optional filename. Only one file may be named. The options are:

-b`type`

Specifies which logical page body lines are to be numbered. Recognised *types* and their meaning are:

- a** number all lines;
- t** number lines with printable text only;
- n** no line numbering;

p`string`

number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is *t* (text lines numbered).

-h`type`

Same as **-b`type`** except for header. Default *type* for logical page header is *n* (no lines numbered).

- ftype** Same as *-btype* except for footer. Default for logical page footer is *n* (no lines numbered).
- p** Do not restart numbering at logical page delimiters.
- vstartnum**
The initial value used to number logical page lines. Default is 1.
- iincr** The increment value used to number logical page lines. Default is 1.
- ssep** The character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
- wwidth**
The number of characters to be used for the line number. Default *width* is 6.
- nformat**
The line numbering format. Recognised values are: *ln*, left justified, leading zeros suppressed; *rn*, right justified, leading zeros suppressed; *rz*, right justified, leading zeros kept. Default *format* is *rn* (right justified).
- lnum**
The number of blank lines to be considered as one. For example, *-l2* results in only the second adjacent blank being numbered (if the appropriate *-ha*, *-ba* and/or *-fa* option is set). Default is 1.
- dxx** The delimiter characters specifying the start of a logical page section may be changed from the default characters "\:" to two user-specified characters. If only one character is entered, the second character remains the default character ":". No space should appear between the *-d* and the delimiter characters. Care should be taken to escape characters that have special meaning to the command interpreter.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number *file1* starting at line number 10 with an increment of 10. The logical page delimiter is "!".

SEE ALSO

pr.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

nm – print name list of object file (DEVELOPMENT)

SYNOPSIS

nm [options] file ...

DESCRIPTION

The *nm* utility displays the symbol table of each object file *file*. The argument *file* may be a relocatable or absolute object file, or it may be an archive of relocatable or absolute object files. For each symbol, at least the following information will be printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address, depending on its storage class.

Size Its size in bytes, if available.

The output of *nm* may be controlled using the following options:

PI -o Print the value and size of a symbol in octal instead of decimal.

UN -x Print the value and size of a symbol in hexadecimal instead of decimal.

-e Print only external and static symbols.

-f Produce full output. Print redundant symbols (*.text*, *.data* and *.bss*), normally suppressed.

-u Print undefined symbols only.

MV UN-V Print the version of the *nm* utility executing on the standard error output.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arg ...*]

DESCRIPTION

The *nohup* utility executes *command* with the signals SIGHUP and SIGQUIT ignored. If output is not redirected by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to *\$HOME/nohup.out*.

Optionally, arguments may be passed to the command by placing them as separate words after the command name.

APPLICATION USAGE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file; this procedure can then be executed as *command*, and the *nohup* applies to everything in the file.

The *nohup* utility is typically used to prevent a program from being killed when the user logs out.

It should be noted that on some implementations of the system, logging out may destroy the environment in which the process is running. On such systems, *nohup* will not necessarily cause the process to continue after the user has logged out.

SEE ALSO

sh, Volume 2, XSI System Interfaces and Headers, *signal()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

od – octal dump

SYNOPSIS

OF **od [-bcdosx] [file] [[+] offset[.][b]]**

DESCRIPTION

The *od* utility writes *file* to the standard output in one or more formats as selected by the options. If no file is specified, the standard input is used. If no option is specified, the output will be in unsigned octal.

For the purposes of this description, *word* refers to a two-byte unit.

The meanings of the options are:

- b Interpret bytes in octal.
- c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: NUL=\0, BS=\b, FF=\f, NL=\n, CR=\r, HT=\t; others appear as 3-digit octal numbers.
- d Interpret *words* in unsigned decimal.
- o Interpret *words* in unsigned octal.
- PI -s Interpret *words* in signed decimal.
- x Interpret *words* in unsigned hexadecimal.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If “.” is appended, the offset is interpreted in decimal. If *b* is appended, the offset is interpreted in units of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

OF **pack** [-] [-f] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

pack

The *pack* utility attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates and owner as those of *name*. The option *-f* will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

OF If the *-* argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency and the code for the byte to be printed on the standard output. Additional occurrences of *-* in place of *name* will cause the internal flag to be set and reset.

The *pack* utility exits with an exit status that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed;
- the filename has more than {NAME_MAX}-2 characters;
- the file has links;
- the file is a directory;
- the file cannot be opened;
- the file is empty;
- no disk storage will be saved by packing;
- a file called *name.z* already exists;
- the *.z* file cannot be created; or
- an I/O error occurred during processing.

The last segment of the filename must contain no more than {NAME_MAX}-2 characters to allow space for the appended *.z* extension.

pcat

The *pcat* utility does for packed files what *cat* does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named **name.z** use:

```
pcat name.z  
or  
pcat name
```

To make an unpacked copy, called **abc**, of a packed file named **name.z** (without destroying **name.z**) use:

```
pcat name >abc
```

The *pcat* utility returns the number of files it was unable to unpack. Failure may occur if:

- the filename (exclusive of the **.z**) has more than (NAME_MAX)-2 characters;
- the file cannot be opened; or
- the file does not appear to be the output of *pack*.

unpack

The *unpack* utility expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates and owner as those of the packed file.

The *unpack* utility exits with an exit status that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

- a file with the "unpacked" name already exists, or
- the unpacked file cannot be created.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree may form the first part of each **.z** file, it is usually not worthwhile to pack small files, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60–75% of their original size. Object files, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Packed files are not necessarily portable to other systems.

SEE ALSO

cat.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

```
paste file ...  
paste -dlist file ...  
paste -s [-dlist] file ...
```

DESCRIPTION

In the first two forms, the *paste* utility concatenates corresponding lines of the given input files. The filename *-* means standard input. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). In the last form above (*-s* option), *paste* combines subsequent lines of the input file (serial merging).

In all cases, lines are glued together with the tab character, unless the *-d* option is used (see below).

Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if *-* is used in place of a filename.

The meanings of the options are:

- dlist* Without this option, the newline characters of each but the last file (or last line in case of the *-s* option) are replaced by a tab character. One or more characters immediately following *-d* replace the default tab as the line-concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no *-s* option) the lines from the last file are always terminated with a newline character, not from the list. The list may contain the special escape sequences: *\n* (newline), *\t* (tab), ** (backslash), and *\0* empty string, not a null character). Quoting may be necessary if characters have special meaning to the command interpreter.
- s* Merge subsequent lines rather than one from each input file. Use a tab for concatenation, unless a list is specified with the *-d* option. Regardless of the list, the very last character of the file is forced to be a newline.

EXAMPLES

1. The following example lists the contents of the current directory in four columns:

```
ls | paste - - - -
```

2. The following example combines pairs of lines into lines:

```
paste -s -d"\t\t" file
```

SEE ALSO

cut, *grep*, *pr*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

pg – file perusal filter for soft-copy terminals

SYNOPSIS

PI **pg** [**-number**] [**-p string**] [**-cefn**s] [**+linenumber**] [**+re/**] [**file ...**]

DESCRIPTION

The *pg* utility is a filter that allows the examination of the named *file* or files one screenful at a time on a soft-copy terminal. (The filename – and/or null arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This utility is different from previous paginators in that it allows the user to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* may optionally attempt to search a system-dependent database, keyed on the value of the *TERM* environment variable. If no information is available, a dumb terminal is assumed.

The options are:

-number

An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23.)

-p string

Cause *pg* to use *string* as the prompt. If the prompt string contains a *%d*, the first occurrence of *%d* in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is “:”.

-c

Home the cursor and clear the screen before displaying each page. This option is ignored if the terminal does not support this.

-e

Cause *pg* not to pause at the end of each file.

-f

Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The **-f** option inhibits *pg* from splitting lines.

UN

-n

Normally, commands must be terminated by a newline character. This option causes an automatic end of command as soon as a command letter is entered.

-s

Cause *pg* to print all messages and prompts in standout mode (usually inverse video) if the terminal supports this.

+linenumber

Start up at *linenumber*.

+re/ Start up at the first line containing the regular expression *re*.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding address; an optionally signed number indicating the point from which further text should be displayed. This address is interpreted in either pages or lines depending on the command. A signed address specifies a point relative to the current page or line, and an unsigned address specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)newline or space

This causes one page to be displayed. The address is specified in pages.

(+1)l With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, by the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1)d or ^D

Simulate scrolling half a screen forward or backward.

The following perusal commands take no address:

. or ^L

Typing a single period causes the current page of text to be redisplayed.

\$ Display the last windowful in the file. Use with caution when the input is a pipe.

IN The following commands are available for searching for text patterns in the text. Simple regular expression syntax is used in patterns. **On systems supporting an internationalised environment, simple internationalised regular expression syntax is used (see Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions).** The *res* must always be terminated by a <newline>, even if the *-n* option is specified.

i/re/ Search forwards for the *i*th occurrence of *re* (default *i*=1). Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^re^

i?re? Search backwards for the *i*th occurrence of *re* (default *i*=1). Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. (The ^ notation is useful for terminals that do not properly handle the ?.)

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending *m* or *b* to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix *t* can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

i n Begin perusing the *i*th next file in the command line. The *i* is an unsigned number; default value is 1.

i p Begin perusing the *i*th previous file in the command line. The *i* is an unsigned number; default is 1.

i w Display another window of text. If *i* is present, set the window size to *i*.

s filename

Save the input in the file named *filename*. Only the current file being perused is saved. The white space between the *s* and *filename* is optional. This command must always be terminated by a newline, even if the *-n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or *Q*

Quit *pg*.

!command

The argument *command* is passed to the command interpreter, whose name is taken from the *SHELL* environment variable. If this is not available, the default command interpreter is used. This command must always be terminated by a newline, even if the *-n* option is specified.

At any time when output is being sent to the terminal, receipt of a quit or interrupt signal causes *pg* to stop sending output and to display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the signal occurs.

If the standard output is not a terminal, *pg* acts just like the *cat* utility, except that a header is printed before each file (if there is more than one).

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes.

If either *LC_COLLATE* or *LC_CTYPE* are not set in the environment or are set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

While waiting for terminal input, *pg* responds to SIGINT and SIGQUIT signals by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These signals should be used with

caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

If terminal tabs are not set every eight positions, undesirable results may occur.

When *pg* is used as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

The *-n* option relies on the *termios* (Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface) terminal interface. If *pg* is being used over an alternative interface, the *-n* option may not be supplied.

SEE ALSO

grep, Volume 2, XSI System Interfaces and Headers, *termios*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

pr – print files

SYNOPSIS

pr [options] [file ...]

DESCRIPTION

The *pr* utility prints *file* or files on the standard output. If *file* is *-*, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, the size of the page being system dependent, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the *-s* option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The options below may appear singly or may be combined in any order:

- PI
- +k** Begin printing with page *k* (default is 1).
 - k** Produce *k*-column output (default is 1). This option should not be used with *-m*. The options *-e* and *-i* are assumed for multi-column output.
 - a** Print multi-column output across the page. This option is appropriate only with the *-k* option.
 - m** Merge and print all files simultaneously, one per column (overrides the *-k* option).
 - d** Double-space the output.
 - eck** Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
 - ick** In *output*, replace spaces wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the horizontal tab character).
 - nck** Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of *-m* output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a horizontal tab).
 - wk** Set the width of a line to *k* character positions for multi-column output (default is 72 for equal-width, multi-column output; no limit otherwise).
 - ok** Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

- lk Set the length of a page to *k* lines (default is system dependent). If *k* is less than what is needed for the page header and trailer, then the option *-t* is in effect; that is, header and trailer lines are suppressed in order to make room for text.
- h *header*
Use *header* as the header to be printed instead of the filename.
- p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).
- f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r Print no diagnostic reports on failure to open files.
- t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a horizontal tab).

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN *LC_TIME* determines the format of date and time strings output in page headers.

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

LC_CTYPE is used to determine if a character is printable. Non-printable characters are still placed on the standard output, but are not counted for the purposes of column-width and line-length calculations.

EXAMPLES

1. Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

2. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ... :

```
pr -e9 -t <file1 >file2
```

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

prs – print an SCCS file (**DEVELOPMENT**)

SYNOPSIS

prs [options] file ...

DESCRIPTION

The *prs* utility prints, on the standard output, parts or all of an SCCS file in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with *s.*) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of options and filenames.

All the described options apply independently to each named file.

–d[*dataspec*]

Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see **Data Keywords**) interspersed with optional user-supplied text.

–r[*SID*]

Used to specify the SCCS identification string (SID) of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

–e Request information for all deltas created earlier than and including the delta designated via the –r option or the date given by the –c option.

–l Request information for all deltas created later than and including the delta designated via the –r option or the date given by the –c option.

UN

–c[*date-time*]

The cutoff *date-time* is in the form:

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values; for example, –c7502 is equivalent to –c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date in the form: –c77/2/2 9:22:25.

–a Request printing of information for both removed, i.e., delta type = *R* (see *rm del*) and existing, i.e., delta type = *D*, deltas. If the –a option is not specified, information for existing deltas only is provided.

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognised data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either simple (S), in which keyword substitution is direct, or multi-line (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognised data keywords. A tab is specified by `\t` and carriage-return-newline is specified by `\n`. The default data keywords are:

:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:

SCCS File Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	:R::L::B::S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th::Tm::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl./excl./ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS: ...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS: ...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS: ...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M

SCCS File Data Keywords

Keyword	Data Item	File Section	Value	Format
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes or no	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of <i>what</i> string	N/A	:Z::M:\t:I:	S
:A:	A form of <i>what</i> string	N/A	:Z::Y::M::I::Z:	S
:Z:	<i>what</i> string delimiter	N/A	@(#)	S
:F:	SCCS filename	N/A	text	S
:PN:	SCCS file pathname	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

1. The following example:

```
prs -d"User Names for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
User Names for s.file are:
xyz
131
abc
```

2. The following example:

```
prs -d"Delta for pgm :M:: :I: - :D: By :P:" -r s.file
```

may produce on the standard output:

```
Delta for pgm main.c: 3.7 - 77/12/1 By cas
```

3. As a special case:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000

MRs:

bl78-12345

bl79-54321

COMMENTS:

this is the comment line for s.file initial delta

for each delta table entry of the *D* type. The only option allowed to be used with this special case is the *-a* option.

SEE ALSO

admin, delta, get, what.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

ps – report process status

SYNOPSIS

OF PI
UN ps [options]

DESCRIPTION

The *ps* utility prints certain information about active processes. Without options, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time and the command name. Otherwise, the information that is displayed is controlled by the selection of options.

The options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The options are:

- e Print information about all processes.
- d Print information about all processes, except session leaders.
- a Print information about all processes, except process group leaders and processes not associated with a terminal.
- f Generate a full listing. (See below for meaning of columns in a full listing.)
- l Generate a long listing. (See below.)
- n *namelist*
The argument will be taken as the name of an alternate system *namelist* file in place of the default.
- t *termlist*
Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's filename (e.g., *tty04*) or if the device's filename starts with *tty*, just the digit identifier (e.g., *04*).
- p *proclist*
Restrict listing to data about processes whose process ID numbers are given in *proclist*.
- u *uidlist*
Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the *-f* option is used, in which case the login name will be printed.
- g *grplist*
Restrict listing to data about processes whose session leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters *f* and *l* (below) indicate the option (*full* or *long*) that causes the corresponding heading to appear; *all* means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F	(l)	Flags (octal and additive) associated with the process.
S	(l)	The state of the process.
UID	(f,l)	The user ID number of the process owner; the login name is printed under the <i>-f</i> option.
PID	(all)	The process ID of the process; it is possible to kill a process if this datum is known.
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilisation for scheduling.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR	(l)	The memory address of the process.
SZ	(l)	The size in blocks of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
STIME	(f)	Starting time of the process.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name; the full command name and its arguments are printed under the <i>-f</i> option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked *defunct*.

Under the option *-f*, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the option *-f*, is printed in square brackets.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN *LC_TIME* determines the format of time strings displayed with the *-f* option (STIME).

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

Things can change while *ps* is running; the snap-shot it gives is only true for an instant, and may not be accurate by the time it is displayed.

On some systems, some of the information described here may not be available, and will consequently not be presented.

SEE ALSO

kill, nice.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

pwd – print working directory name

SYNOPSIS

pwd

DESCRIPTION

The *pwd* command writes the pathname of the working (current) directory to the standard output.

This command operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

This is usually a shell built-in command.

SEE ALSO

cd, **Volume 2, XSI System Interfaces and Headers**, *getcwd()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this command in an 8-bit transparent manner has been noted.

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [-fri] file ...

rmdir dir ...

DESCRIPTION

rm The *rm* utility removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with *y*, or the locale's equivalent of a *y*, the file is deleted, otherwise the file remains. No questions are asked when the option *-f* is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument *-r* has been used. In that case, *rm* recursively deletes the entire contents of the specified directory and the directory itself.

If the option *-i* (interactive) is in effect, *rm* asks whether to delete each file and, under *-r*, whether to examine each directory.

rmdir

The *rmdir* utility removes entries for the named directories, which must be empty.

These utilities operate in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

Environment Variables

IN *LANG* determines the locale's equivalent of *y* (for yes/no queries).

If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

It is forbidden to remove the *file ...* in order to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

SEE ALSO

Volume 2, XSI System Interfaces and Headers, *remove()*, *rmdir()*, *unlink()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of these utilities in an 8-bit transparent manner has been noted.

The operation of these utilities in an internationalised environment has been described.

NAME

rmidel – remove a delta from an SCCS file (**DEVELOPMENT**)

SYNOPSIS

rmidel -rSID file ...

DESCRIPTION

The *rmidel* utility removes the delta specified by the SID from each named SCCS file. The delta to be removed must be the most recent delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must not be that of a version being edited for the purpose of making a delta (i.e., if a *p-file*, see *get*, exists for the named SCCS file, the SID specified must not appear in any entry of the *p-file*).

If a directory is named, *rmidel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with *s.*), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

Removal of a delta is restricted to: (1) the user who made the delta; (2) the owner of the SCCS file; (3) the owner of the directory containing the SCCS file.

SEE ALSO

delta, *get*, *prs*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

sact – print current SCCS file-editing activity (**DEVELOPMENT**)

SYNOPSIS

sact file...

DESCRIPTION

The *sact* utility informs the user of any impending deltas to a named SCCS file. This situation occurs when *get -e* has been executed previously without a subsequent execution of *delta*. If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces:

- Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
- Field 2 specifies the SID for the new delta to be created.
- Field 3 contains the login name of the user who will make the delta (i.e., who executed a *get* for editing).
- Field 4 contains the date that *get -e* was executed.
- Field 5 contains the time that *get -e* was executed.

SEE ALSO

delta, get, unget.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

sdb – symbolic debugger (**DEVELOPMENT**) (**OPTIONAL**)

SYNOPSIS

OP UN **sdb** [*objfile*] [*corfile*] [*directory-list*]

DESCRIPTION

The *sdb* utility is a symbolic debugger that can be used with C-language programs, and possibly programs in other languages. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

The argument *objfile* is an executable program file which has been compiled with the *-g* (debug) option; if it has not been compiled with the *-g* option the symbolic capabilities of *sdb* will be limited, but the file can still be examined and the program debugged. The default for *objfile* is **a.out**. The argument *corfile* is assumed to be a core image file produced after executing *objfile*; the default for *corfile* is **core**. The core file need not be present. A *-* in place of *corfile* will force *sdb* to ignore any core image file. The colon-separated list of directories (*directory-list*) is used to locate the source files used to build *objfile*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfile* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main()*. The current line and file may be changed with the source file examination commands.

Warnings are provided if the source files used in producing *objfile* cannot be found, or are newer than *objfile*.

Names of variables are written just as they are in C-language. (The *sdb* utility does not truncate names.) Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *pointer->member* and array elements as *variable[number]*. Pointers may be dereferenced by using the form *pointer[0]*. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value that *sdb* displays, not the addresses of individual elements.

Elements of a multi-dimensional array may be referenced as *variable[number][number]...*, or as *variable[number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, *** may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures,

sdb displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C-language may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as *filename:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or filename is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program.

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable/clm

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b one byte
h two bytes (half word)
l four bytes (long word)

Legal values for *m* are: ,

c character
d decimal
u decimal, unsigned
o octal
x hexadecimal
s assume *variable* is a string pointer and print characters starting at the address pointed to by the variable
a print characters starting at the variable's address. This format may not be used with register variables
p pointer to procedure
i disassemble machine-language instruction with addresses printed numerically and symbolically

The length specifiers are only effective with the formats *c*, *d*, *u*, *o* and *x*. Any of the specifiers, *c*, *l* and *m*, may be omitted. If all are omitted, *sdb* chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the variable. If a count specifier is used for the *s* or *a* commands, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *"/"*.

linenumber?lm

variable:?lm

Print the value at the address from *a.out* or I-space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is *i*.

variable=lm

linenumber=lm

number=lm

Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then *lx* is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!value

Set *variable* to the given *value*. The *value* may be a number, a character constant or a variable. The *value* must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted *'character'*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C-language conventions are used in any type conversions necessary to perform the indicated assignment.

x Print the machine registers and the current machine-language instruction.

The commands for examining source files are:

e *procedure*
e *filename*
e *directory/*
e *directory filename*

The first two forms set the current file to the file containing *procedure* or to *filename*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, filename or directory is given, the current procedure name and filename are reported.

/*regular expression/*

Search forward from the current line for a line containing a string matching *regular expression*. Simple regular-expression syntax may be used. The trailing / may be omitted.

?regular expression?

Search backward from the current line for a line containing a string matching *regular expression*. Simple regular expression syntax may be used. The trailing ? may be omitted.

p Print the current line.

z Print the current line followed by the next nine lines. Set the current line to the last line printed.

w Window. Print the ten lines around the current line.

number

Set the current line to the given line number. Print the new current line.

The commands for controlling the execution of the source program are:

count r *args*

count R

Run the program with the given arguments. The *r* command with no arguments reuses the previous arguments to the program while the *R* command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber c *count*

linenumber C *count*

Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. With the *C* command, the signal which caused the program to stop is reactivated; with the *c* command, it is ignored. If a line number is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes. (It may not be possible to set breakpoints in some places, e.g., with shared libraries.)

s count

S *count*

Single step the program through *count* lines. If no *count* is given then the program is run for one line. *S* is equivalent to *s* except it steps through procedure calls.

i

I Single step by one machine-language instruction. With the *I* command, the signal which caused the program to stop is reactivated; with the *i* command, it is ignored.

k Kill the program being debugged.

procedure(arg1,arg2,...)

procedure(arg1,arg2,...)/m

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to *d*.

linenumber b commands

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., *proc:*), a breakpoint is placed at the first line in the procedure even if it was not compiled with the *-g* option. If no *linenumber* is given, a breakpoint is placed at the current line. (It may not be possible to set breakpoints in some places, e.g., with shared libraries.) If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If *k* is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

B Print a list of the currently active breakpoints.

linenumber d

Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a *y* or *d*, then the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

Miscellaneous commands:

!command

The command is interpreted by the command interpreter.

<newline>

If the previous command printed a source line, then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

<EOF>

Scroll. Print the next ten lines of instructions, source or data depending on which was printed last.

< filename

Read commands from *filename* until the end-of-file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; *<* may not appear as a command in a file.

" string

Print the given string. The C-language escape sequences of the form *\character* are recognised, where *character* is a non-numeric character.

q Exit the debugger.

APPLICATION USAGE

By their very nature, symbolic debuggers are strongly system dependent. Although *sdb* is optional in the Guide, strong efforts will be made to provide it, or similar functionality. Application developers should look at the documentation for their current system; common alternatives are *adb*, *dbx* and *cdb*.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

sed – stream editor

SYNOPSIS

sed [-n] [-e script] [-f sfile] [file ...]

DESCRIPTION

The *sed* utility copies the named files (standard input default) to the standard output, edited according to a script of commands. The *-f* option causes the script to be taken from file *sfile*; these options accumulate. If there is just one *-e* option and no *-f* options, the flag *-e* may be omitted. The *-n* option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a *D* command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under *-n*) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the pattern space for subsequent retrieval.

An address is either a decimal number that counts input lines cumulatively across files, a \$ that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of the *ed* command modified as follows:

- In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address *\xabc\xdefx*, the second *x* stands for itself, so that the regular expression is *abcxdef*.
- The escape sequence *\n* matches a newline embedded in the pattern space.
- A *“.”* matches any character except the terminal newline of the pattern space.
- A command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address, to the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

IN The *sed* utility makes use of simple regular expression syntax. On systems supporting an internationalised environment, simple internationalised regular expression syntax is used (see Volume 3, XSI Internationalisation, Chapter 6, Regular Expressions).

Editing commands can be applied only to non-selected pattern spaces by use of the negation function *!* (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The argument *text* consists of one or more lines, all but the last of which end with \ to hide the newline. Backslashes in *text* are treated like backslashes in the replacement string of an *s* command, and may be used to protect initial spaces and tabs against the stripping that is done on every script line. The argument *rfile* or the argument *wfile* must terminate the command line and must be preceded by exactly one space. Each *wfile* is created before processing begins. There can be at most ten distinct *wfile* arguments.

- (1)a\
 - text* Append. Place *text* on the output before reading the next input line.
- (2)b *label*
 - Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.
- (2)c\
 - text* Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
- (2)d Delete the pattern space. Start the next cycle.
- (2)D Delete the initial segment of the pattern space through the first newline. Start the next cycle.
- (2)g Replace the contents of the pattern space by the contents of the hold space.
- (2)G Append the contents of the hold space to the pattern space.
- (2)h Replace the contents of the hold space by the contents of the pattern space.
- (2)H Append the contents of the pattern space to the hold space.
- (1)i\
 - text* Insert. Place *text* on the standard output.
- OF (2)l List the pattern space on the standard output in an unambiguous form. Non-printing characters are displayed in a (hopefully) mnemonic form, and long lines are folded.
- (2)n Copy the pattern space to the standard output if the default output has not been suppressed (by the *-n* option or the *#n* command in the script file). Replace the pattern space with the next line of input.
- (2)N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)
- (2)p Print. Copy the pattern space to the standard output.
- (2)P Copy the initial segment of the pattern space through the first newline to the standard output.
- (1)q Quit. Branch to the end of the script. Do not start a new cycle.
- (2)r *rfile*
 - Read the contents of *rfile*. Place them on the output before reading the next input line.

(2)s/regular expression/replacement/flags

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see the *s* command of *ed*. The value of *flags* is zero or more of:

- n** *n* = 1–512. Substitute for just the *n*th occurrence of the *regular expression*.
- g** Global. Substitute for all non-overlapping instances of the *regular expression* rather than just the first one.
- p** Print the pattern space if a replacement was made and the default output has been suppressed (via the *-n* option on the command line or the *#n* command in the script file).

w wfile

Write. Append the pattern space to *wfile* if a replacement was made.

(2)t label

Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a *t*. If *label* is empty, branch to the end of the script.

(2)w wfile

Write. Append the pattern space to *wfile*.

(2)x Exchange the contents of the pattern and hold spaces.**(2)y/string1/string2/**

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! function

Don't. Apply the *function* (or group, if *function* is "{") only to lines not selected by the address(es).

(0): label

This command does nothing; it bears a *label* to which *b* and *t* commands can branch.

(1)= Place the current line number on the standard output as a line.**(2){** Execute the following commands through a matching "}" only when the pattern space is selected.**(0)** An empty command is ignored.**(0)#** If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an *n*, then the default output will be suppressed. The rest of the line after *#n* is also ignored. A script file must contain at least one non-comment line.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed regular expressions, *LC_COLLATE* determines the behaviour of range expressions, equivalence classes and multi-character collating symbols, and *LC_CTYPE* determines the behaviour of character classes.

LC_CTYPE determines which characters are classified as non-printing for the *l* (list) command.

If either *LC_COLLATE* or *LC_CTYPE* are not set in the environment or are set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

This utility supports internationalised regular expressions, see **Volume 3, XSI Internationalisation**.

SEE ALSO

awk, ed, grep.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The regular expressions have been marked as being of the internationalised type on systems supporting an internationalised environment.

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

sh – shell, the command interpreter

SYNOPSIS

sh [flags] [arg ...]

DESCRIPTION

The *sh* utility is a command interpreter that executes commands read from a terminal or a file. The command *sh -r* offers a restricted version of the command interpreter; it is used to set up login names and execution environments whose capabilities are more controlled. See **Invocation**, below, for the meaning of flags and other arguments to the shell.

Definitions

A *blank* is a tab or a space.

A *name* is a sequence of letters, digits or underscores beginning with a letter or underscore.

A *parameter* is a name, a digit or any of the characters *, @, #, ?, -, \$ and !.

A <tab> is the horizontal tab character ('\t').

A <space> is the space character (' ').

A <newline> is the newline character ('\n').

Commands

A *simple command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (in **Volume 2, XSI System Interfaces and Headers**, see *exec*). The *value* of a *simple command* is its exit status if it terminates normally, or (octal) 0200+status if it terminates abnormally. See **APPLICATION USAGE**, below, for a list of status values.

A *pipeline* is a sequence of one or more commands separated by |. The standard output of each command but the last is connected by a pipe, see *pipe()*, to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The *exit status* of a *pipeline* is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, && or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. The symbol ; causes sequential execution of the preceding pipeline; the symbol & causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a list, instead of semicolons, to delimit commands.

A *command* is either a *simple command* or one of the following. Unless otherwise stated, the *value* returned by a *command* is that of the last *simple command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a *for* command is executed, *name* is set to the next *word* taken from the *in word* list. If *in word* ... is omitted, then the *for* command executes the *do* list once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...] *list* ;;] ... **esac**

A *case* command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see **File Name Generation**) except that a slash, a leading dot or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following *if* is executed and, if it returns a zero exit status, the *list* following the first *then* is executed. Otherwise, the *list* following *elif* is executed and, if its value is zero, the *list* following the next *then* is executed. Failing that, the *else* list is executed. If no *else* list or *then* list is executed, the *if* command returns a zero exit status.

while *list* **do** *list* **done**

A *while* command repeatedly executes the *while* list and, if the exit status of the last command in the list is zero, executes the *do* list; otherwise the loop terminates. If no commands in the *do* list are executed, then the *while* command returns a zero exit status; *until* may be used in place of *while* to negate the loop termination test.

(*list* **)**

Execute *list* in a subshell.

{ *list* **;**

list is simply executed. (The semicolon may be replaced by a newline.)

name() { *list* ; }

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. (The semicolon may be replaced by a newline.) Execution of functions is described below (see **Execution**).

The following words are only recognised as the first word of a command and when not quoted:

```
if then else elif fi case
esac for while until do done { }
```

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of back quotes (``) may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character `$` is used to introduce substitutable parameters. There are two types of parameter; positional and keyword. If the parameter name is a single digit [0–9], it is a positional parameter; otherwise, the name must be a valid name as defined above, and gives a keyword parameter. Positional parameters may be assigned values by *set*. Keyword parameters (also known as variables) may be assigned values by writing:

```
name=value [ name=value ]...
```

Pattern matching is not performed on *value*. There cannot be a function and a variable with the same name.

`${parameter}`

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit or underscore that is not to be interpreted as part of its name. If *parameter* is `*` or `@`, all the positional parameters, starting with `$1`, are substituted (separated by spaces). Parameter `$0` is set from argument zero when the shell is invoked.

`${parameter:-word}`

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

`${parameter:=word}`

If *parameter* is not set or is null, set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

`${parameter:?word}`

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message *parameter null or not set* is printed.

`${parameter:+word}`

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string so that, in the following example, *pwd* is executed only if *d* is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon, `:`, is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- `#` The number of positional parameters in decimal.
- `-` Flags supplied to the shell on invocation or by the *set* command.
- `?` The decimal value returned by the last synchronously executed command.
- `$` The process number of this shell.
- `!` The process number of the last background command invoked.

The following parameters are used by the shell:

- CDPATH** The search path for the *cd* command. The syntax and usage are similar to that of *PATH*.

IFS	Internal field separators, normally <space>, <tab>, and <newline>.
HOME	The default argument (home directory) for the <i>cd</i> command.
MAIL	If this parameter is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file. The user is informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set.
MAILCHECK	This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> parameters. The default value is 600 seconds. If set to 0, the shell will check before each primary prompt.
MAILPATH	A colon-separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by % and a message that will be printed when the modification time changes.
PATH	The search path for commands (see <i>Execution</i> below). The user may not change <i>PATH</i> if executing in restricted mode.
PS1	Primary prompt string, by default \$.
PS2	Secondary prompt string, by default >.
SHACCT	WITHDRAWN.
SHELL	When the shell is invoked, it scans the environment (see <i>Environment</i> below) for this name. If it is found and there is an <i>r</i> in the filename part of its value, the shell becomes a restricted shell.
LANG	If this parameter specifies a valid locale name, it is used to initialise the shell's international environment. <i>LANG</i> identifies general localisation requirements for language, cultural data and coded character set.
LC_COLLATE	Identifies specific localisation requirements for text collation.
LC_CTYPE	Identifies specific localisation requirements for character classification and case conversion.
LC_MONETARY	Identifies specific localisation requirements for formatting monetary values.
LC_NUMERIC	Identifies specific localisation requirements for number formats.
LC_TIME	Identifies specific localisation requirements for date and time formats.
NLSPATH	The search path for locating native-language message catalogues. This is used internally to locate message catalogues containing shell messages.

Changes to the international environment variables (i.e., *LANG*, *LC_COLLATE*, *LC_CTYPE*, *LC_MONETARY*, *LC_NUMERIC* and *LC_TIME*) take immediate effect.

The shell gives default values to *IFS*, *MAILCHECK*, *PATH*, *PS1* and *PS2*.

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in *IFS*) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the shell metanotation characters *, ? and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with a sorted list of filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character "." at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & () | ^ < > <newline> <space> <tab>

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \<newline> is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, `, " and \$. "\$*" expands to the positional parameters, starting from 1. When the expansion occurs within a double-quoted string, it expands to a single argument. Positional parameters are separated by the first character of the *IFS*. Whereas "\$@" expands to the positional parameters, starting from 1. When the expansion occurs within a double-quoted string, it expands to as many arguments as there are positional parameters.

Prompting

When used interactively, the shell prompts with the value of *PS1* before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of *PS2*) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simplecommand* or may precede or follow a command and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*

Use file *word* as standard input (file descriptor 0).

>*word*

Use file *word* as standard output (file descriptor 1). If the file does not exist it is

created; otherwise it is truncated to zero length.

>>*word*

Use file *word* as standard output. If the file exists output is appended to it; otherwise the file is created.

<<[-] *word*

The shell input is read up to a line that is the same as *word* or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) "`\<newline>`" is ignored, and `\` must be used to quote the characters `\`, `$`, ``` and the first character of *word*. If `-` is appended to `<<`, all leading tabs are stripped from *word* and from the document.

<&*digit*

Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using `>&digit`.

<&- The standard input is closed. Similarly for the standard output using `>&-`.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left to right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with the file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by `&` the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* is a list of name–value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the *export* command is used to bind the shell's parameter to the environment (see also *set -a*). A parameter may be removed from the environment with the *unset* command. The environment seen by any executed command is thus

composed of any unmodified name–value pairs originally inherited by the shell, minus any pairs removed by *unset*, plus any modifications or additions, all of which must be noted in *export* commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=123 cmd
(export TERM; TERM=123; cmd)
```

(where *cmd* uses the value of the environment variable *TERM*) are equivalent as far as the execution of *cmd* is concerned.

If the *-k* flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. The following first prints *a=b c* and *c*:

```
echo a=b c
set -k
echo a=b c
```

Signals

The SIGINT and SIGQUIT signals for an invoked command are ignored if the command is followed by *&*; otherwise signals have the values inherited by the shell from its parent (but see also the *trap* command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the **Special Commands** listed below, it is executed in the shell process. If the command name does not match a special command, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters *\$1*, *\$2*, ... are set to the arguments of the function. If the command name matches neither a special command nor the name of a defined function, a new process is created and an attempt is made to execute the command via one of the *exec* functions, see **Volume 2, XSI System Interfaces and Headers**, *exec*.

The variable *PATH* defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between two colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory from left to right in the path is searched in turn for an executable file of the name of the command. If the file has execute permission but is not an executable file, it is assumed to be a file containing shell commands. A subshell is spawned to read it. A parenthesised command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *exec* calls later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the *PATH* variable is changed or the *hash -r* command is executed (see below).

Special Commands

- :** Null effect; the command does nothing. Arguments are parsed as usual. A zero exit code is returned.
- . *file***
Read and execute commands from *file* and return. The search path specified by *PATH* is used to find the directory containing *file*.
- break [*n*]**
Exit from the enclosing *for* or *while* loop, if any. If *n* is specified break *n* levels.
- cd [*arg*]**
Change the current directory to *arg*. The variable *HOME* is the default *arg*. The variable *CDPATH* defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is *null* (specifying the current directory). Note that the current directory is specified by a null pathname, which can appear immediately after the equals sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg* and the name of the directory selected may be printed. The *cd* command may not be executed in restricted mode.
- continue [*n*]**
Resume the next iteration of the enclosing *for* or *while* loop. If *n* is specified resume at the *n*th enclosing loop.
- echo [*arg* ...]**
Echo arguments. See *echo* for usage and description.
- eval [*arg* ...]**
The arguments are read as input to the shell and the resulting command(s) executed.
- exec [*arg* ...]**
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- exit [*n*]**
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit).
- export [*name* ...]**
The given *names* are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may not be exported.

hash [*-r* | *name ...*]

For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The *-r* option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented.

pwd Print the current working directory. See *pwd* for usage and description.

read [*name ...*]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Only the characters in the variable *IFS* are recognised as delimiters. The return code is 0 unless an end-of-file is encountered.

readonly [*name ...*]

The given *names* are marked *readonly* and the values of these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]

Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [*--aefhknrtuvx* [*arg ...*]]

- a* Mark variables which are modified or created for export.
- e* Exit immediately if a command exits with a non-zero exit status.
- f* Disable filename generation.
- h* Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- k* All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- n* Read commands but do not execute them.
- t* Exit after reading and executing one command.
- u* Treat unset variables as errors when substituting.
- v* Print shell input lines as they are read.
- x* Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting *\$1* to *-*.

Using *+* rather than *-* causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in *\$-*. The remaining arguments are positional parameters and are assigned, in order, to *\$1*, *\$2*, If no arguments are given, the values of all names are printed.

shift [*n*]

The positional parameters from $\$n+1$ are renamed $\$1, \$2, \dots$. If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test* for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the empty string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The *trap* command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*-f n*]

If the *-f n* option is used, then a size limit of *n* 512-byte units is imposed on files written by the shell and its child processes (files of any size may be read). Only a process with appropriate privileges may increase the limit. If *n* is omitted, the current limit is printed. If no option is given, *-f* is assumed.

umask [*ooo*]

The user file-creation mask is set to the octal value *ooo*, see **Volume 2, XSI System Interfaces and Headers**, *umask()*. If *ooo* is omitted, the current value of the mask is printed.

unset [*name* ...]

For each *name*, remove the corresponding variable or function. The variables *PATH*, *PS1*, *PS2*, *MAILCHECK* and *IFS* cannot be unset.

wait [*n*]

Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through one of the *exec* functions (see **Volume 2, XSI System Interfaces and Headers**, *exec*), and the first character of argument zero is *-*, commands are initially read from */etc/profile* and from *\$HOME/.profile*, if such files exist. Thereafter, commands are read as described below. The flags below are interpreted by the shell on invocation only; note that unless the *-c* or *-s* is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string*

If the **-c** flag is present commands are read from *string*.

-s

If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for special commands) is written to file descriptor 2.

-i

If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case SIGTERM is ignored (so that *kill 0* does not kill an interactive shell) and SIGINT is caught and ignored (so that *wait* is interruptible). In all cases, SIGQUIT is ignored by the shell.

-r

If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the *set* command above.

Restricted Shell

A restricted shell is used to set up login names and execution environments whose capabilities are more controlled. In a restricted shell, the following are disallowed:

- changing directory, see *cd*
- setting the value of *PATH*
- specifying command names containing /
- redirecting output (> and >>)

The restrictions above are enforced after *.profile* is interpreted.

When a command to be executed is found to be a shell procedure, an unrestricted shell is invoked to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the *.profile* has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within filename generation patterns, *LC_COLLATE* determines the behaviour of range expressions, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes. *LC_CTYPE* also determines which characters are defined as *letters* in the current coded character set (character class *alpha*).

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the *exit* command, above).

APPLICATION USAGE

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. The *hash* command should be used to correct this situation.

If the current directory or the one above it is moved, *pwd* may not give the correct response. The *cd* command with a full pathname should be used to correct this situation.

The output from the built-in commands should not be relied upon for use by other programs, as its format is not rigorously specified.

The following numeric values for the signals can be used in the *trap* utility:

1	SIGHUP
2	SIGINT
3	SIGQUIT
9	SIGKILL
15	SIGTERM

Refer to **Volume 2, XSI System Interfaces and Headers**, *signal()* for the definitions of these signals.

If *sh* is invoked by the name *rsh*, typically via a link to *sh*, it behaves in the restricted way described under the *-r* option. However, the name *rsh* is used for a command in common use in networking extensions, which has different behaviour. For this reason, the description of *rsh* has been removed.

IN In an international environment, character ordering is determined by the setting of *LC_COLLATE*, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particularly when using range expressions in filename generation patterns. For example, the command,

```
rm [a-z]*
```

might be expected to match all filenames beginning with a lower-case alphabetic character. However, if dictionary ordering was identified in *LC_COLLATE*, it would also match all filenames beginning with an upper-case character. Conversely, it would fail to match letters collated after 'z' in languages such as Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form,

```
rm [[:lower:]]*
```

This uses `LC_CTYPE` to determine character classes and will work predictably for all supported languages and codesets. For shell scripts produced on non-internationalised systems (or without consideration for the above dangers), it is recommended that they be executed in a native machine environment. This requires that `LANG`, `LC_COLLATE`, etc. should be set to the null string, or should not be defined at all.

Any valid letter can be used in a *name* (as defined by category `LC_CTYPE`), although for portability to non-internationalised systems it is recommended that only 7-bit characters are used in command names. The order in which filename lists are sorted is defined by the setting of `LC_COLLATE`.

SEE ALSO

cd, *echo*, *pwd*, *test*, *umask*, **Volume 2, XSI System Interfaces and Headers**, *dup()*, *exec*, *exit()*, *fork()*, *pipe()*, *signal()*, *system()*, *ulimit()*, *umask()*, *wait()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

sleep

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

The *sleep* utility suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command) &
```

or to execute a command every so often, as in:

```
while true
do
    command
    sleep 37
done
```

This utility operates in an 8-bit transparent manner, see Section 2.2.1, Eight Bit Transparent Utilities.

SEE ALSO

Volume 2, XSI System Interfaces and Headers, *alarm()*, *sleep()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

sort – sort and/or merge files

SYNOPSIS

PI **sort** [**-cmu**] [**-ooutput**] [**-y[kmem]**] [**-zrecsz**] [**-dfnr**] [**-btx**] [**+pos1**] [**-pos2**]
[**file...**]

DESCRIPTION

The *sort* utility sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **-** is used as a filename or no input files are named.

IN Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is either lexicographic by bytes in machine-collating sequence, or, in an internationalised environment, ordering is defined by collating sequence information in the program's environment.

The following options alter the default behaviour:

- c** Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.
- m** Merge only, the input files are already sorted.
- u** Unique: suppress all but one in each set of lines having equal keys.

-ooutput

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **-o** and *output*.

PI **-y[kmem]**

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **-y0** is guaranteed to start with minimum memory. By convention, **-y** (with no argument) starts with maximum memory.

-zrecsz

The size of the longest line read in the sort phase is recorded so that buffers of the correct size can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a system-dependent default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules:

- d Only alphanumeric, space and tab characters are significant in comparisons.
- f Fold case of letters. Prior to being compared, all letters are effectively folded into one case. In the "C" locale, a letter is folded as if by a call to the *toupper()* function. In any other locale, the folding algorithm is implementation defined.
- i Ignore all non-printing characters during non-numeric comparisons.
- n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional radix character, is sorted by arithmetic value. The *-n* option implies the *-b* option (see below). Note that the *-b* option is only effective when restricted sort-key specifications are in effect.
- r Reverse the sense of comparisons.

When ordering options appear before restricted sort-key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a *field*, a minimal sequence of characters followed by a field separator or a newline. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the following options:

- tx Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).
- b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the *-b* option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the *b* flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

The arguments *pos1* and *pos2* each have the form *m.n* optionally followed by one or more of the flags *bdfinr*. A starting position specified by *+m.n* is interpreted to mean the (*n*+1)th character in the (*m*+1)th field. A missing *.n* means .0, indicating the first character of the (*m*+1)th field. If the *b* flag is in effect, *n* is counted from the first non-blank in the (*m*+1)th field; *+m.0b* refers to the first non-blank character in the (*m*+1)th field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the *b* flag is in effect *n* is counted from the last leading blank in the (*m*+1)th field; *-m.1b* refers to the first non-blank in the (*m*+1)th

field.

IN Within restricted sort-key specifications, a "character" is defined as any collating symbol. Thus skip over multi-character collating elements is performed automatically.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN `LC_COLLATE` determines the default ordering rules applied to the sort; `LC_CTYPE` determines the behaviour of character classification for the `-d`, `-f` and `-i` options; `LC_NUMERIC` determines the definition of the radix character for the `-n` option.

If either `LC_COLLATE`, `LC_CTYPE` or `LC_NUMERIC` are not set in the environment or are set to the empty string, the value of `LANG` will be used as a default for each unset/empty variable. If `LANG` is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXIT STATUS

The *sort* utility comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the `-c` option.

When the last line of an input file is missing a newline character, *sort* appends one, prints a warning message and continues.

EXAMPLES

1. Sort the contents of **infile** with the second field as the sort key:
`sort +1 -2 infile`
2. Sort, in reverse order, the contents of **infile1** and **infile2**, placing the output in *outfile* and using the first character of the second field as the sort key:
`sort -r -o outfile +1.0 -1.2 infile1 infile2`
3. Sort, in reverse order, the contents of **infile1** and **infile2** using the first non-blank character of the second field as the sort key:
`sort -r +1.0b -1.1b infile1 infile2`

4. Print the lines of the already sorted file **infile**, suppressing all but the first occurrence of lines having the same third field (the options *-um* with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

APPLICATION USAGE

The user is warned that the semantics of the *+pos1 -pos2* are different in earlier versions of *sort*.

The *-f* option may not work as expected in locales where there is not a one-to-one mapping between an upper- and a lower-case letter.

SEE ALSO

comm, *join*, *uniq*, **Volume 2, XSI System Interfaces and Headers**, *toupper()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

spell – find spelling errors

SYNOPSIS

OF LA **spell** [-v] [-b] [-x] [+local_file] [file ...]

DESCRIPTION

The *spell* utility collects words from the named files and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes and/or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Under the *-v* option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the *-b* option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*.

Under the *-x* option, every plausible stem is printed with = for each word.

Under the *+local_file* option, words found in *local_file* are removed from *spell*'s output. The argument *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

FUTURE DIRECTIONS

In order to conform to the command syntax standard, the *+local_file* option will be changed to the form *-flocal_file*. The old form will continue to be accepted for some time.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

split

Utilities

NAME

split – split a file into pieces

SYNOPSIS

split [-n] [file [name]]

DESCRIPTION

The *split* utility reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with *aa* appended, and so on lexicographically, up to *zz* (a maximum of 676 files). The argument *name* cannot be longer than {NAME_MAX}-2 characters. If no output name is given, x is default.

If no input file is given, or if - is given in its stead, then the standard input is used.

SEE ALSO

csplit.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

strip – strip symbolic information from an object file (DEVELOPMENT)

SYNOPSIS

MV UN strip [-x] [-r] [-V] file ...

DESCRIPTION

The *strip* utility strips the symbolic information from object files or archives of object files.

The amount of information stripped from the symbol-table can be controlled by using any of the following options:

MV UN -x Do not strip static or external symbol information.

MV UN -r Do not strip static or external symbol information, or relocation information.

MV UN -V Print the version of the *strip* command on the standard error output.

If there is any relocation information in the object file and any symbol-table information is to be stripped, *strip* will report an error and terminate without stripping *file* unless the *-r* flag is used.

APPLICATION USAGE

The purpose of this command is to reduce the file storage overhead taken by the object file.

SEE ALSO

ar, *cc*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

stty – set the options for a terminal

SYNOPSIS

UN **stty [-a] [-g] [options]**

DESCRIPTION

The *stty* utility sets certain terminal I/O options for the device that is its standard input; without arguments, it reports the settings of certain options; with the *-a* option, it reports all of the option settings; with the *-g* option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in **Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (-parenb)

enable (disable) parity generation and detection.

parodd (-parodd)

select odd (even) parity.

cs5 cs6 cs7 cs8

select character size.

0 hang up line immediately. This applies to all terminal lines, not just modem lines. A SIGHUP signal is sent to all processes attached to the line.

number

set terminal baud rate to the *number* given, if possible. (Not all speeds are supported by all hardware interfaces.)

hupcl (-hupcl)

hang up (do not hang up) modem connection on last close.

hup (-hup)

same as *hupcl* (-*hupcl*).

cstopb (-cstopb)

use two (one) stop bits per character.

cread (-cread)

enable (disable) the receiver.

clocal (-clocal)

assume a line without (with) modem control.

loblk (-loblk)

block (do not block) output from a non-current layer. The *loblk* option is supported on systems that support *shl*. This option may not be supported by all hardware interfaces.

tostop (-tostop)

Block (do not block) output from background job. This option may not be supported by all hardware interfaces.

Input Modes**ignbrk (-ignbrk)**

ignore (do not ignore) break on input.

brkint (-brkint)

signal (do not signal) SIGINT on break.

ignpar (-ignpar)

ignore (do not ignore) parity errors.

parmrk (-parmrk)

mark (do not mark) parity errors.

inpck (-inpck)

enable (disable) input parity checking.

istrip (-istrip)

strip (do not strip) input characters to seven bits.

inlcr (-inlcr)

map (do not map) newline to carriage-return on input.

igncr (-igncr)

ignore (do not ignore) carriage-return on input.

icrnl (-icrnl)

map (do not map) carriage-return to newline on input.

iuclic (-iuclic)

map (do not map) upper-case alphabets to lower-case on input.

ixon (-ixon)

enable (disable) START/STOP output control.

ixany (-ixany)

allow any character to restart output.

ixoff (-ixoff)

request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes**opost (-opost)**

post-process output (do not post-process output; ignore all other output modes).

olcuc (-olcuc)

map (do not map) lower-case alphabets to upper-case on output.

onlcr (-onlcr)

map (do not map) newline to carriage-return-newline on output.

- ocrnl** (**-ocrnl**)
map (do not map) carriage-return to newline on output.
- onocr** (**-onocr**)
do not (do) output carriage-return at column zero.
- onlret** (**-onlret**)
the terminal newline key performs (does not perform) the carriage-return function.
- ofill** (**-ofill**)
use fill characters (use timing) for delays.
- ofdel** (**-ofdel**)
fill characters are DELs (NULs).
- cr0 cr1 cr2 cr3**
select style of delay for carriage-returns.
- nl0 nl1**
select style of delay for line-feeds.
- tab0 tab1 tab2 tab3**
select style of delay for horizontal tabs.
- bs0 bs1**
select style of delay for backspaces.
- ff0 ff1**
select style of delay for form-feeds.
- vt0 vt1**
select style of delay for vertical tabs.

Local Modes

- isig** (**-isig**)
enable (disable) the checking of characters against the special control characters INTR and QUIT.
- icanon** (**-icanon**)
enable (disable) canonical input (ERASE and KILL processing).
- xcase** (**-xcase**)
canonical (unprocessed) upper/lower-case presentation.
- echo** (**-echo**)
echo back (do not echo back) every character typed.
- echoe** (**-echoe**)
echo (do not echo) ERASE character as a backspace-space-backspace string.
Note: this mode will erase the ERASEed character on many CRT terminals; however, it does not keep track of column position and, as a result, may be confusing on escaped characters, tabs and backspaces.

echok (**-echok**)

echo (do not echo) newline after KILL character.

echonl (**-echonl**)

echo (do not echo) newline.

noflsh (**-noflsh**)

disable (enable) flush after INTR or QUIT.

Control Assignments

control-character c

OP set *control-character* to *c*, where *control-character* is *erase*, *kill*, *intr*, *quit*, *switch*, *eof*, *eol*, *min* or *time* to represent the characters ERASE, KILL, INTR, QUIT, EOF, EOL, MIN or TIME respectively (MIN and TIME are used with *-icanon*). If *c* is preceded by a caret (^), then the value used is the corresponding CTRL character (e.g., "**^D**" is a CTRL-D); "**^?**" is interpreted as DEL and "**^-**" is interpreted as undefined.

line i

set line discipline to *i* in the range [0, 127].

Combination Modes**evenp** or **parity**

enable *parenb* and *cs7*.

oddp

enable *parenb*, *cs7* and *parodd*.

-parity, **-evenp** or **-oddp**

disable *parenb*, and set *cs8*.

raw (**-raw** or **cooked**)

enable (disable) raw input and output (equivalent to *cs8*, with no ERASE, KILL, INTR, QUIT, EOF, EOL or output post-processing, and no parity).

nl (**-nl**)

unset (set) *icrnl*, *onlcr*. In addition **-nl** unsets *inlcr*, *igncr*, *ocrnl* and *onlret*.

lcase (**-lcase**)

set (unset) *xcase*, *iuclic* and *olcuc*.

LCASE (**-LCASE**)

same as *lcase* (**-lcase**).

tabs (**-tabs** or **tab8**)

preserve tabs (expand to spaces) when printing.

ek

reset ERASE and KILL characters back to the system defaults.

sane

reset all modes to some reasonable values.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Typical implementations of this utility require a communications line configured to use the *termios* interface (see **Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**). On systems where none of these lines are available, this utility may not be present.

SEE ALSO

Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

sum – print checksum and block count of a file

SYNOPSIS

OF PI **sum** [-r] [file ...]

DESCRIPTION

UN The *sum* utility calculates and prints a checksum for the named file and also prints the space used by the file, in 512-byte units. The option *-r* causes an alternative algorithm to be used in computing the checksum. If no files are named, the standard input is read.

APPLICATION USAGE

It is not clear that the algorithms used in typical implementations are portable, i.e., the same checksum may not be produced for the same input on different systems.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

tabs – set tabs on a terminal

SYNOPSIS

MV UN **tabs** [**tabspec**] [**+mn**] [**-Ttype**]

DESCRIPTION

The *tabs* utility sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings.

Three types of tab specification are accepted for *tabspec*; "canned", repetitive or arbitrary. If no *tabspec* is given, the default value is *-8*, i.e., "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0.

The options below give the set of "canned" formats:

- a** 1,10,16,36,72
Assembler, applicable to some mainframes.
- a2** 1,10,16,40,72
Assembler, applicable to some mainframes.
- c** 1,8,12,16,20,55
COBOL, normal format.
- c2** 1,6,10,14,49
COBOL, compact format (columns 1–6 omitted).
- c3** 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1–6 omitted), with more tabs than *-c2*.
- f** 1,7,11,15,19,23
FORTRAN
- p** 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I
- s** 1,10,55
SNOBOL
- u** 1,12,20,44
Assembler, applicable to some mainframes.

In addition to these "canned" formats, three other types exist:

- n** A repetitive specification requests tabs at columns $1+n$, $1+2*n$, etc. Of particular importance is the value *-8*; this represents the "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value *-0*, implying no tabs at all.

n1,n2,...

The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type*

The *tabs* utility usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. The argument *type* is a terminal name. If no **-T** flag is supplied, *tabs* searches for the environment variable *TERM*. If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

APPLICATION USAGE

This makes use of the terminal's hardware tabs, and also the *stty tabs* option.

This utility is not recommended for application use. It is a strong candidate for future removal.

SEE ALSO

stty.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [\pm [*number*][*lbc*[*f*]]] [*file*]

DESCRIPTION

The *tail* utility copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning or *-number* from the end of the input (if *number* is not given, the value 10 is assumed). If neither *+* nor *-* is given then *-* is assumed. The argument *number* is counted in units of lines, 512-byte units, or characters, according to the appended option *l*, *b* or *c*. When no units are specified, counting is by lines.

With the *f* (follow) option, if the input file is not a pipe, the program will not terminate after the last line of the input file has been copied, but will enter an endless loop: it sleeps for at least a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Tails relative to the end of the file are saved in a buffer, and thus are limited in length.

Various kinds of anomalous behaviour may occur with character special files.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

tar – file archiver

SYNOPSIS

tar [option] [file ...]

DESCRIPTION

The *tar* utility creates archives of files. Its actions are controlled by the *option* argument. The *option* is a string of characters containing at most one function letter and possibly one or more modifiers. Other arguments to the command are *file* or files (or directory names) specifying which files are to be archived or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the option is specified by one of the following letters:

- r** The named *file* or files are written on the end of the archive.
- x** The named *file* or files are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. If a named file in the archive does not exist on the system, the file is created with the same mode as the one in the archive, except that the set-user-ID and set-group-ID modes are not set unless the user has appropriate privileges. If the files exist, their modes are not changed except as described above. The owner, group, and modification time are restored (if possible). If no *files* argument is given, the entire content of the archive is extracted. Note that if several files with the same name are in the archive, the last one overwrites all earlier ones.
- t** The names of all the files in the archive are listed.
- u** The named *file* or files are added to the archive if they are not already there, or have been modified since last written into the archive.
- c** Create a new archive; writing begins at the beginning of the archive, instead of after the last file.

The following characters may be used in addition to the letter that selects the desired function:

- v** Normally, *tar* does its work silently. The *v* (verbose) modifier causes it to type the name of each file it treats, preceded by the option letter. With the *t* option, *v* gives more information about the archive entries than just the name.
- w** Causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with *y*, or the locale's equivalent of *y*, is given, the action is performed. Any other input means "no". This modifier is invalid with the *t* option.

IN

- f** Causes *tar* to use the next argument as the name of the archive instead of the system-dependent default. If the name of the file is `-`, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. The *tar* utility can also be used to move directory hierarchies with the command:

```
(cd fromdir; tar cf - .) | (cd todir; tar xf -)
```
- b** Causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with (raw) magnetic tape archives (see *f* above). The block size is determined automatically when reading tapes (options *x* and *t*).
- l** Tells *tar* to report if it cannot resolve all of the links to the files being archived. If *l* is not specified, no error messages are printed. This modifier is valid only with the options *c*, *r* and *u*.
- m** Tells *tar* not to restore the modification times. The modification time of the file will be the time of extraction. This modifier is invalid with the *t* option.
- PI o** Causes extracted files to take on the user and group identifier of the user running the program rather than those on the archive. This modifier is valid only with the *x* option.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

- IN LC_TIME** determines the format of date and time strings output when listing the contents of an archive with the *v* option, e.g.,

```
tar tvf /dev/tape
```

LANG determines the locale's equivalent of *y* (for yes/no queries). If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

ERRORS

The *tar* utility reports bad option characters and read/write errors. It also reports an error if enough memory is not available to hold the link tables.

APPLICATION USAGE

Volume 3, XSI Source Code Transfer discusses use of *tar* between X/Open systems.

The *r* and *u* options may not function correctly if the archive being manipulated is a magnetic tape device.

Refer to **Volume 3, XSI Source Code Transfer** for guidelines on portability of source code between X/Open systems.

IN **ISO 8859-1:1987** is defined as the transmission codeset for communication between X/Open systems. However, 8-bit data and filenames may not be portable to non-internationalised systems. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines, and that only characters defined in the Portable Filename Character Set are used for naming files.

FUTURE DIRECTIONS

The **IEEE P1003.2** standard is currently proposing an alternative interface to data interchange media. It is possible that this utility will be withdrawn in a future Issue if the functionality described here is subsumed into the new utility.

SEE ALSO

cpio.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The behaviour of this utility in an internationalised environment has been described.

NAME

tee – join pipes and make copies of input

SYNOPSIS

tee [-i] [-a] [file...]

DESCRIPTION

The *tee* utility transcribes the standard input to the standard output and makes copies in the *file* or files named. The *-i* option ignores interrupts; the *-a* option causes the output to be appended to the *file* or files rather than overwriting them.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

test – condition evaluation command

SYNOPSIS

test *expr*

[*expr*]

DESCRIPTION

The *test* command evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

- r *file* true if *file* exists and is readable.
- w *file* true if *file* exists and is writable.
- x *file* true if *file* exists and is executable.
- f *file* true if *file* exists and is a regular file.
- d *file* true if *file* exists and is a directory.
- c *file* true if *file* exists and is a character special file.
- b *file* true if *file* exists and is a block special file.
- p *file* true if *file* exists and is a named pipe (FIFO).
- u *file* true if *file* exists and its set-user-ID bit is set.
- g *file* true if *file* exists and its set-group-ID bit is set.
- s *file* true if *file* exists and has a size greater than zero.
- t [*fildev*] true if the open file whose file descriptor number is *fildev* (1 by default) is associated with a terminal device.
- z *s1* true if the length of string *s1* is zero.
- n *s1* true if the length of the string *s1* is non-zero.
- s1* = *s2* true if strings *s1* and *s2* are identical.
- s1* != *s2* true if strings *s1* and *s2* are not identical.
- s1* true if *s1* is not the empty string.
- n1* -eq *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons *-ne*, *-gt*, *-ge*, *-lt* and *-le* may be used in place of *-eq*.

These primaries may be combined with the following operators:

- !** unary negation operator.
- a** binary *and* operator.
- o** binary *or* operator (*-a* has higher precedence than *-o*).
- (*expr*)** parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses may be meaningful to the command interpreter and therefore may need to be escaped.

In the second form of the command (i.e., the one that uses `[]`, rather than the word *test*), the square brackets must be delimited by blanks.

This command operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

This is usually a shell built-in command.

SEE ALSO

find, sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this command in an 8-bit transparent manner has been noted.

NAME

time – time a command

SYNOPSIS

time command [arg...]

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent executing system code, and the time spent in execution of the user code. Times are reported in seconds.

The times are printed on standard error.

Optionally, *arguments* may be passed to the command by placing them as separate words after the command name.

APPLICATION USAGE

When *time* is used on a multi-processor system the sum of system and user time could be greater than real time.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

touch – update access and modification times of a file

SYNOPSIS

UN touch [-amc] [mmddhhmm[yy]] file ...

DESCRIPTION

The *touch* utility causes the access and modification times of each *file* to be updated. The *file* is created if it does not exist. The *-a* and *-m* options cause touch to update only the access or modification times respectively (default is *-am*). The *-c* option silently prevents *touch* from creating the *file* if it did not previously exist.

UN If a time is specified, that time is used in place of the current time.

EXIT STATUS

The exit status from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

APPLICATION USAGE

Completely numeric filenames may cause confusion as *touch* may assume this is a date argument.

FUTURE DIRECTIONS

Because of the parsing ambiguity introduced by the date format marked as UN, a *-d date* option will be introduced to support this functionality.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

tr – transliterate characters

SYNOPSIS

tr [-cds] [string1 [string2]]

DESCRIPTION

The *tr* utility copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options *-cds* may be used:

- c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
- d Deletes all input characters in *string1*.
- s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[*c-c*] Stands for the ordered string of collating elements *c-c*, inclusive.

UN

[[:*class*:]]

Stands for the string of characters in type *class*. Which must be one of *alpha*, *upper*, *lower*, *digit*, *xdigit*, *alnum*, *space*, *punct*, *print*, *graph* or *cntrl*. Character classes are expanded in collation order.

[[:*c*=]]

Stands for an equivalence class, i.e. any character defined as having the same relative order in the current collation sequence as *c*.

[[:*cc*:]]

Stands for a collating symbol. Multi-character collating elements must be represented as collating symbols to distinguish them from single character collating elements.

[*a*n*] Stands for *n* repetitions of *a*. If the first digit of *n* is 0, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used to remove special meaning from any character following it in a string. In addition, \ followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN Within bracketed expressions, *LC_COLLATE* determines the behaviour of ranges, equivalence classes and multi-character collating elements, and *LC_CTYPE* determines the behaviour of character classes. *LC_COLLATE* also determines the order in which character class expressions are expanded. *LC_CTYPE* determines which characters are defined in the current universe of characters when using the *-c* option.

If either *LC_COLLATE* or *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default for each unset/empty variable. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

EXAMPLES

1. The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabets. The strings are quoted to protect the special characters from interpretation by the *command* interpreter; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

- IN
2. The following example achieves the same results in an internationalised environment and works for any supported codeset.

```
tr -cs "[[:alpha:]]" "[\012*]" <file1 >file2
```

3. The next example may translate all lower-case characters in *file1* to upper-case and write the results to standard output.

```
tr "[[:lower:]]" "[[:upper:]]" <file1
```

Note that character classes specified in either *string1* or *string2* are expanded in collation order. As a consequence, this will not work in locales in which there is not a one to one mapping between lower- and upper-case letters, or in which lower- and upper-case letters collate in a different relative order.

4. This example uses an equivalence class to identify accented variants of the base character *e* in *file1*, which are stripped of diacritical marks and written to *file2*.

```
tr "[[=e=]]" e <file1 >file2
```

5. Finally, this example translates instances of the multi-character collating element "ch" to upper-case. Note that the single-character collating elements "c" and "h" will not be affected by this operation unless they are part of a "ch" character sequence.

```
tr "[[.ch.]]" "[[.CH.]]" <file1 >file2
```


APPLICATION USAGE

The *tr* utility does not handle ASCII NUL in *string1* or *string2*; it always deletes NUL from input.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

The *true* utility returns exit code zero. The *false* utility returns a non-zero exit code.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXIT STATUS

Exit status is:

<i>true</i>	zero
<i>false</i>	non-zero

EXAMPLE

This command will be executed forever:

```
while true
do
    command
done
```

APPLICATION USAGE

These utilities are typically used in shell scripts, as shown in the **EXAMPLE**.

SEE ALSO

sh.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

tsort – topological sort

SYNOPSIS

tsort [*file*]

DESCRIPTION

The *tsort* utility produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is used.

The input consists of pairs of items (non-empty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [-s]

DESCRIPTION

The *tty* utility prints the pathname of the user's terminal. The *-s* option inhibits printing of the terminal pathname, allowing one to test just the exit code.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

EXIT STATUS

Exit codes:

- 0 if standard input is a terminal.
- 1 otherwise.
- 2 if invalid options were specified.

An error is reported if the standard input is not a terminal and *-s* is not specified.

APPLICATION USAGE

On systems supporting virtual terminals, *tty* returns the name of the virtual terminal, not the real terminal, when the former is in use.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [*ooo*]

DESCRIPTION

The *umask* command sets the user file-creation mode mask to *ooo*. These three octal digits refer to read/write/execute permissions for *owner*, *group* and *others*, respectively, see *chmod*. The bits which are set to 1 in *ooo* will be zero in the permission field of all subsequently created files.

For example, *umask 022* removes *group* and *others* write permission (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

If *ooo* is omitted, the current value of the mask is printed.

This command operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

This is always a shell built-in command.

SEE ALSO

chmod, **Volume 2, XSI System Interfaces and Headers**, *umask()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this command in an 8-bit transparent manner has been noted.

NAME

uname – print name of current system

SYNOPSIS

OF **uname** [-snrvma]

DESCRIPTION

The *uname* utility prints the current system name on the standard output file. The options cause selected information returned by *uname()* to be printed:

- s print the system name (default). This is a name by which the system is known in the local installation.
- n print the nodename. The nodename may be a name by which the system is known to a communications network.
- r print the operating system release.
- v print the operating system version.
- m print the machine hardware name.

OF -a print all the above information.

OF Combinations of the above options may be used.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

SEE ALSO

uname().

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

unget – undo a previous get of an SCCS file (**DEVELOPMENT**)

SYNOPSIS

unget [-rSID] [-s] [-n] files

DESCRIPTION

The *unget* utility undoes the effect of a *get -e* done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-rSID

Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the new delta.) The use of this option is necessary only if two or more outstanding *gets* for editing on the same SCCS file were done by the same person (login name). An error is reported if the specified SID is ambiguous, or if it is necessary and omitted on the command line.

-s Suppress the printout, on the standard output, of the intended delta's SID.

-n Causes the retention of the file that was obtained by *get*, which would normally be removed from the current directory.

SEE ALSO

delta, get, sact.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [-udc [+n] [-n]] [input [output]]

DESCRIPTION

The *uniq* utility reads the *input* file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the *output* file. If *input* and *output* are omitted, *uniq* reads from standard input and writes to standard output. If *output* is omitted, standard output is assumed. The arguments *input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found, see *sort*. If the *-u* flag is used, just the lines that are not repeated in the original file are output. The *-d* option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the *-u* and *-d* mode outputs.

The *-c* option supersedes *-u* and *-d* and generates an output report with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbours.

+n The first *n* characters are ignored. Fields are skipped before characters.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

IN In an internationalised environment, the value of the *LC_COLLATE* environment variable must be equal to the value it had when the input files were sorted.

If *uniq* does not support selection of collating sequences via *LC_COLLATE*, the input files must be sorted according to the collating sequence of the "C" locale (see **Volume 3, XSI Supplementary Definitions, Chapter 7, C Program Locale**).

SEE ALSO

comm, *sort*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

uucp, uulog, uuname – system-to-system copy

SYNOPSIS

UN **uucp** [options] *source-file* ... *destination-file*

UN **uulog** [-s *system*]

UN **uuname** [-l]

DESCRIPTION**uucp**

The *uucp* utility copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on your machine, or may have the form:

system-name!*pathname*

where *system-name* is taken from a list of system names that *uucp* knows about. The destination *system-name* may also be a list of names such as

system-name!*system-name*!...!*system-name*!*pathname*

in which case, an attempt is made to send the file via the specified route to the destination. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The shell metacharacters *?*, ***, and *[...]* appearing in *pathname* will be expanded on the appropriate system.

Pathnames may be one of:

1. a full pathname.
2. a pathname preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory. Note that if an invalid login is specified, the default will be to the public directory (PUBDIR).
3. a pathname preceded by *~/destination* where *destination* is appended to PUBDIR .

Note: This destination will be treated as a filename unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a */* . For example, *~/dan/* as the destination will make the directory PUBDIR/*dan* if it does not exist and put the requested file(s) in that directory.

4. anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

The *uucp* utility gives universal read and write permissions and preserves execute permissions across the transmission.

The following options are interpreted by *uucp*:

- UN **-c** Do not copy local file to the spool directory for transfer to the remote machine (default).
- UN **-C** Force the copy of local files to the spool directory for transfer.
- UN **-d** Make all necessary directories for the file copy (default).
- UN **-f** Do not make intermediate directories for the file copy.
- UN **-j** Output the job identification string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.
- m** Send mail to the requester when the copy is completed.
- UN **-nuser** Notify *user* on the remote system that a file was sent.
- UN **-r** Do not start the file transfer; just queue the job.

uulog

The *uulog* utility queries a log file of *uucp* or *uux* transactions.

- OF If the **-s** option is specified, then *uulog* prints information about file transfer work involving system *system*.

uname

The *uname* utility lists the *uucp* names of known systems, one per line. The **-l** option returns the local system name.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

- IN **LC_TIME** determines the format of date and time strings output by *uucp* and *uulog*.

If **LC_TIME** is not set in the environment or is set to the empty string, the value of **LANG** will be used as a default. If **LANG** is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

The domain of remotely accessible files can (and for obvious security reasons usually should) be severely restricted.

Typical implementations of this utility require a communications line configured to use the *termios* (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) interface. On systems where none of these lines are available, this utility may not be present.

- IN As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate system. On an internationalised system, this is done under the control of local settings of `LC_COLLATE` and `LC_CTYPE`. Thus, care should be taken when using bracketed filename patterns, as collation and typing rules may vary from one system to another. Also be aware that certain types of expression (i.e., equivalence classes, character classes and collating symbols) may not be supported on non-internationalised systems.
- IN **ISO 8859-1:1987** is defined as the transmission codeset for communication between X/Open systems. However, the 8-bit operation of the `uucp()` utility cannot be guaranteed in all circumstances. For example, transmission data may be restricted to 7-bits by the underlying network, 8-bit data and filenames may not be portable to non-internationalised systems, etc. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines, and that only characters defined in the Portable Filename Character Set are used for naming remote files.

SEE ALSO

mail, uustat, uux.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

UN **uustat** [**options**]

DESCRIPTION

The *uustat* utility will display the status of, or cancel, previously specified *uucp* requests, or provide general status on *uucp* connections to other systems.

Not all combinations of *options* are valid. Only one of the following *options* can be specified with *uustat*:

UN **-q** List the jobs queued for each machine.

-k *jobid*

Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person invoking *uustat* unless that user has appropriate privileges.

UN **-r** *jobid*

Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup program from deleting the job until the jobs modification time reaches the limit imposed by the program.

The *options* below may not be used with the ones listed above; however, these *options* may be used singly or together:

-s *sys*

Report the status of all *uucp* requests for remote system *sys*.

-u *user*

Report the status of all *uucp* requests issued by *user*.

When no *options* are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN **LC_TIME** determines the format of date and time strings output by *uustat*.

If **LC_TIME** is not set in the environment or is set to the empty string, the value of **LANG** will be used as a default. If **LANG** is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

APPLICATION USAGE

Typical implementations of this utility require a communications line configured to use the *termios* (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) interface. On systems where none of these lines are available, this utility may not be present.

SEE ALSO

uucp.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

uuto, uupick – public system-to-system file copy

SYNOPSIS

UN **uuto** [-p] [-m] source-file ... destination

UN **uupick** [-s system]

DESCRIPTION

uuto

The *uuto* utility sends *source-files* to *destination*. The *uuto* utility uses the *uucp* facility to send files, while it allows the local system to control the file access. A *source-file* name is a pathname on the user's machine. Destination has the form: *system!user* where *system* is taken from a list of system names that *uucp* knows about, see *uname* in *uucp*. The argument *user* is the login name of someone on the specified system.

Two options are available:

-p Copy the source file into the spool directory before transmission.

-m Send mail to the sender when the copy is complete.

The files (or subtrees if directories are specified) are sent to a public directory (PUBDIR) on *system*. Specifically, the files are sent to the directory

PUBDIR/receive/user/fsystem

where *user* is the recipient, and *fsystem* is the sending system.

The recipient is notified by *mail* of the arrival of files.

uupick

The *uupick* utility may be used by a user to accept or reject the files transmitted to the user. Specifically, *uupick* searches PUBDIR on the user's system for files sent to the user. For each entry (file or directory) found, one of the following messages is printed on the standard output:

from system: dir dirname ?

from system: file filename ?

The *uupick* utility then reads a line from the standard input to determine the disposition of the file. The user's possible responses are:

<newline>

Go on to next entry.

d Delete the entry.

m [*dir*]

Move the entry to named directory *dir*. If *dir* is not specified as a complete pathname a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.

a [*dir*]

Same as *m* except moving all the files sent from *system*.

p Print the content of the file to standard output.

q Stop and exit.

<EOF>

Same as *q*.

!command

Escape to the command interpreter to execute *command*.

***** Print a usage summary for *uuto*.

The *uupick* utility invoked with the *-s system* option will only search for files (and list any found) sent from *system*.

These utilities operate in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Typical implementations of this utility require a communications line configured to use the *termios* (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) interface. On systems where none of these lines are available, this utility may not be present.

IN **ISO 8859-1:1987** is defined as the transmission codeset for communication between X/Open systems. However, the 8-bit operation of the *uuto()* utility cannot be guaranteed in all circumstances. For example, transmission data may be restricted to 7-bits by the underlying network, 8-bit data and filenames may not be portable to non-internationalised systems, etc. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines, and that only characters defined in the Portable Filename Character Set are used for naming remote files.

SEE ALSO

mail, uucp, uustat, uux.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

uux – remote command execution

SYNOPSIS

UN **uux [options] command-string**

DESCRIPTION

The *uux* utility will gather zero or more files from various systems, execute a command on a specified system, and then send the standard output of the command to a file on a specified system.

The *command-string* is made up of one or more arguments that are similar to normal command arguments, except that the command and any filenames may be prefixed by "*system-name!*". A null *system-name* is interpreted as the local system.

The following statements are relevant if *sh* is the command interpreter.

The metacharacter *** will not give the desired result.

The redirection tokens *>>* and *<<* are not implemented.

A filename may be specified as for *uucp*; it may be a full pathname, a pathname preceded by *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest* (*dest* is prefixed by PUBDIR), or a simple filename (which is prefixed by the current directory). See *uucp* for the details.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the *file1* and *file2* files from the *usg* and *pwba* machines, execute *diff*, and put the results in *file.diff* in the local PUBDIR/*dan* directory. (PUBDIR is the *uucp* public directory on the local system.)

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the non-local filenames (without path or machine reference) must be unique within the *uux* request.

The following command will not work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

because the file *xyz* will be copied from the *b* system as well as the *c* system, causing a name conflict. The command:

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work (provided *diff* is a permitted command), because the local file *xyz* (which is not copied) does not conflict with the copied file *xyz* from the *c* system.

Any characters special to the command interpreter should be quoted either by quoting the entire *command-string* or quoting the special characters as individual arguments.

The command *uux* will attempt to get all files to the execution system. For files that are output files, the filename must be escaped using parentheses. For example, the command:

```
uux a!cut -f1 b!/usr/file \ (c!/usr/file\)
```

gets */usr/file* from system *b*, sends it to system *a*, performs a *cut* command on that file, and sends the result of the *cut* command to system *c*.

The command *uux* will notify the user (by mail) if the requested command on the remote system was disallowed. This notification can be turned off by the *-n* option. The response comes by mail from the remote machine.

The following *options* are interpreted by *uux*:

- The standard input to *uux* is made the standard input to the *command-string*.
- UN -j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.
- n Do not notify the user if the command fails.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail via *uux*.

Only the first command of a pipeline, see *sh*, may have a *system-name!*. All other commands are executed on the system of the first command.

Typical implementations of this utility require a communications line configured to use the *termios* (**Volume 2, XSI System Interfaces and Headers, Section 2.6, General Terminal Interface**) interface. On systems where none of these lines are available, this utility may not be present.

IN As noted in *uucp*, shell metacharacters appearing in pathnames are expanded on the appropriate local system. On an internationalised system, this is done under the control of local settings of *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename patterns, as collation and typing rules may vary from one system to another. Also be aware that certain types of expression (i.e., equivalence classes, character classes and collating symbols) may not be supported on non-internationalised systems.

IN ISO 8859-1:1987 is defined as the transmission codeset for communication between X/Open systems. However, the 8-bit operation of the *uux* utility cannot be guaranteed in all circumstances. For example, transmission data may be restricted to 7-bits by the underlying network, 8-bit data and filenames may not be portable to non-internationalised systems, etc.. Under these circumstances, it is recommended that only characters defined in the ASCII 7-bit range of characters are used for data transfer between machines, and that only characters defined in the Portable Filename Character Set are used for naming remote files.

SEE ALSO

uucp, uustat.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

val – validate SCCS file (**DEVELOPMENT**)

SYNOPSIS

val –
val [-s] [-rSID] [-mname] [-ytype] file ...

DESCRIPTION

The *val* utility determines if the specified *file* is an SCCS file meeting the characteristics specified by the options. The arguments may appear in any order.

The *val* utility has a special argument, “-”, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command-line argument list.

The *val* utility generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The options are defined as follows. The effects of any option apply independently to each named file on the command line.

- s Silence the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
- rSID *SID* (SCCS Identification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e.g., -r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e.g., -r1.0 or -r1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.
- mname
name is compared with the SCCS %M% keyword in *file*.
- ytype
type is compared with the SCCS %Y% keyword in *file*.

EXIT STATUS

The 8-bit code returned by *val* is a disjunction of the possible errors, i.e., it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument;
- bit 1 = unknown or duplicate option;
- bit 2 = corrupted SCCS file;
- bit 3 = cannot open file or file not SCCS;
- bit 4 = *SID* is invalid or ambiguous;
- bit 5 = *SID* does not exist;
- bit 6 = %Y%, -y mismatch;
- bit 7 = %M%, -m mismatch.

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned: a logical *or* of the codes generated for each command line and file processed.

SEE ALSO

admin, delta, get, prs.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

vi – screen-oriented (visual) display editor

SYNOPSIS

OP `vi [-r] [-l] [-wn] [-R] [+command] [file ...]`

DESCRIPTION

The *vi* (visual) utility is a display-oriented text editor. It is based on the underlying line editor *ex*: it is possible to switch back and forth between the two and to execute *ex* commands from within *vi*.

When using *vi*, the terminal screen acts as window into the file being edited. Changes made to the file are reflected in the screen display; the position of the cursor on the screen indicates the position within the file.

The environment variable *TERM* must give the terminal type; the terminal must be defined in a terminal description database. As for *ex*, editor initialisation scripts can be placed in the environment variable *EXINIT* or the file *.exrc* in the current or home directory.

Options

The following options are interpreted by *vi*:

-r Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files will be printed.

-l Set lisp mode (see **Edit Options** in *ex*).

-wn Set the default window size to *n*.

-R Read-only mode; the read-only flag is set, preventing accidental overwriting of the file.

+command

The specified *ex command* is interpreted before editing begins.

Vi Commands**General Remarks**

See *ex* for the complete description of *ex*. Only the visual mode of the editor is described here.

At the beginning, *vi* is in the command mode; the input mode is entered by several commands used to insert or change text. In input mode, ESC (escape) is used to leave input mode; in command mode, it is used to cancel a partial command; the terminal bell is sounded if the editor is not in input mode and there is no partially entered command.

The last (bottom) line of the screen is used to echo the input for search commands / and ?, for *ex* commands :, and system commands !. It is also used to report errors or print other messages.

Receipt of SIGINT during text input or during the input of a command on the bottom line, terminates the input (or cancels the command) and returns the editor to command mode. During command mode, SIGINT causes the bell to be sounded; in general the bell indicates an error (such as unrecognised key).

Lines displayed on the screen containing only ~ indicate that the last line above them is the last line of the file (the ~ lines are past the end of the file). On a terminal with limited local intelligence, there may be lines on the screen marked with an @: these indicate space on the screen not corresponding to lines in the file. (These lines may be removed by entering ^R, forcing the editor to retype the screen without these holes.)

Command Summary

Most commands accept a preceding number as an argument, either to give a size or position (for display or movement commands), or as a repeat count (for commands that change text). For simplicity, this optional argument will be referred to as *count* when its effect is described.

The following operators can be followed by a movement command, in order to specify an extent of text to be affected: *c*, *d*, *y*, *<*, *>*, *!* and *=*. The region specified is from the current cursor position to just before the cursor position indicated by the move. If the command operates on lines only, then all the lines which fall partly or wholly within this region are affected. Otherwise the exact marked region is affected.

In the following, control characters are indicated in the form ^X, which stands for control-X.

Unless otherwise specified, the commands are interpreted in command mode and have no special effect in input mode.

- ^B Scroll backward to display the window above the current one. A count specifies the number of windows to go back. Two lines of overlap are kept if possible.
- ^D Scroll forward a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future ^D and ^U commands.

In input mode, backs *shiftwidth* spaces over the indentation provided by *autoindent* or ^T.
- ^E Scroll forward one line, leaving the cursor where it is if possible.
- ^F Scroll forward to display the window below the current one. A count specifies the number of windows to go forward. Two lines of overlap are kept if possible.
- ^G Print the current filename and other information, including the number of lines and the current position. (Equivalent to the *ex* command *f*.)

- `^H`** Move one space to the left (stops at the left margin). A count specifies the number of spaces to back up. (Same as *h*.)
In input mode, backs over the last input character without erasing it.
- `^J`** Move the cursor down one line in the same column. A count specifies the number of lines to move down. (Same as `^N` and *j*.)
- `^L`** Clear and redraw the screen. (Used when the screen is scrambled for any reason.)
- `^M`** Move to the first non-white character in the next line. A count specifies the number of lines to go forward.
- `^N`** Same as `^J` and *j*.
- `^P`** Move the cursor up one line in the same column. A count specifies the number of lines to move up. (Same as *k*.)
- `^R`** Redraw the current screen, eliminating the false lines marked with @ (which do not correspond to actual lines in the file).
- `^T`** In input mode, if at the beginning of the line or preceded only by white space, inserts *shiftwidth* white space. This inserted space can only be backed over using `^D`.
- `^U`** Scroll up a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future `^D` and `^U` commands.
- `^V`** In input mode, quotes the next character to make it possible to insert special characters (including ESC) into the file.
- `^W`** In input mode, backs up one word; the deleted characters remain on the display.
- `^Y`** Scroll backward one line, leaving the cursor where it is if possible.
- `^I`** Cancel a partially formed command; sounds the bell if there is none.
In input mode, terminates input mode.
When entering a command on the bottom line of the screen (*ex* command line or search pattern with \ or ?), terminates input and executes command.
- `<space>`** Move one space to the right (stops at the end of the line). A count specifies the number of spaces to go forwards. (Same as *l*.)
- `!`** An operator which passes specified lines from the buffer as standard input to the specified system command, and replaces those lines with the standard output from the command. The `!` is followed by a movement command specifying the lines to be passed (lines from the current position to the end of the movement) and then the command (terminated as usual by a return). A count preceding the `!` is passed on to the movement command after `!`.

Doubling ! and preceding it by a count causes that many lines, starting with the current line, to be passed.

" Precedes a named buffer specification. There are named buffers 1-9 in which the editor places deleted text. The named buffers *a-z* are available to the user for saving deleted or yanked text, see also *y*, below.

\$ Move to the end of the current line. A count specifies the number of lines to go forward (e.g., 2\$ goes to the end of the next line).

% Move to the parenthesis or brace which matches the parenthesis or brace at the current cursor position.

& Same as the *ex* command & (repeats previous substitute command).

` When followed by ` , returns to the previous context, placing the cursor at the beginning of the line. (The previous context is set whenever a non-relative move is made.) When followed by a letter *a-z*, returns to the line marked with that letter (see the *m* command), at the first non-white character in the line.

When used with an operator such as *d* to specify an extent of text, the operation takes place over complete lines. (See also `.)

^ When followed by ^ , returns to the previous context, placing the cursor at the character position marked. (The previous context is set whenever a non-relative move is made.) When followed by a letter *a-z*, returns to the line marked with that letter (see the *m* command), at the character position marked.

When used with an operator such as *d* to specify an extent of text, the operation takes place from the exact marked place to the current position within the line. (See also ^.)

[[Back up to the previous section boundary. A section is defined by the value of the *sections* option. Lines which start with a form-feed (^L character) or { also stop [[.

If the *lisp* option is set, stops at each "(" at the beginning of a line.

]] Move forward to a section boundary (see [[).

^ Move to the first non-white position on the current line.

(Move backward to the beginning of a sentence. A sentence ends at a ".", ! or ? followed by either the end of a line or by two spaces. Any number of closing),], " and ` characters may appear between the ".", ! or ? and the spaces or end of line. A count moves back that many sentences.

If the *lisp* option is set, moves to the beginning of a *lisp* s-expression. Sentences also begin at paragraph and section boundaries (see { and [[).

-) Move forward to the beginning of a sentence. A count moves forward that many sentences. (See "'".)
- { Move back to the beginning of the preceding paragraph. A paragraph is defined by the value of the *paragraphs* option. A completely empty line and a section boundary (see [] above) are also taken to begin paragraphs. A count specifies the number of paragraphs to move backward.
- } Move forward to the beginning of the next paragraph. A count specifies the number of paragraphs to move forward. (See {.)
- | Requires a count; the cursor is placed in that column (if possible).
- + Move to the first non-white character in the next line. A count specifies the number of lines to go forward. (Same as ^M.)
- , Reverse of the last *f*, *F*, *t* or *T* command, looking the other way in the current line. A count is equivalent to repeating the search that many times.
- Move to the first non-white character in the previous line. A count specifies how many lines to move back.
- . Repeat the last command which changed the buffer. A count is passed on to the command being repeated.
- / Read a string from the last line on the screen, interpret it as a regular expression, and scan forward for the next occurrence of a matching string. The search begins when carriage-return is entered to terminate the pattern; it may be terminated by sending SIGINT.

When used with an operator to specify an extent of text, the defined region is from the current cursor position to the beginning of the matched string. Whole lines may be specified by giving an offset from the matched line (using a closing / followed by *+n* or *-n*).
- 0 Move to the first character on the current line. (Not interpreted as a command when preceded by a non-zero digit.)
- : Begins an *ex* command. The ":", as well as the entered command, are echoed on the bottom line; it is executed when the input is terminated by entering a return.
- ; Repeat the last single character find using *f*, *F*, *t* or *T*. A count is equivalent to repeating the search that many times.
- < An operator that shifts lines left one *shiftwidth*. May be followed by a move to specify lines. A count is passed through to the move command.

When repeated (<<), shifts the current line (or count lines starting at the current one).
- > An operator that shifts lines right one *shiftwidth*. (See <.)

- = If the *lisp* option is set, reindents the specified lines, as though they were typed in with *lisp* and *autoindent* set. May be preceded by a count to indicate how many lines to process, or followed by a move command for the same purpose.
- ? Scan backwards; the reverse of /. (See /.)
- A Append at the end of line. (Same as \$a.)
- B Back up a word, where a word is any non-blank sequence, placing the cursor at the beginning of the word. A count gives the number of words to go back.
- C Change the rest of the text on the current line. (Same as c\$.)
- D Delete the rest of the text on the current line. (Same as d\$.)
- E Move forward to the end of a word, where a word is any non-blank sequence. A count gives the number of words to go forward.
- F Must be followed by a single character; scan backwards in the current line for that character, moving the cursor to it if found. A count is equivalent to repeating the search that many times.
- G Go to the line number given as preceding argument, or the end of the file if no preceding count is given.
- H Move the cursor to the top line on the screen. If a count is given, the cursor is moved to that line on the screen, counting from the top. The cursor is placed on the first non-white character on the line. If used as the target of an operator, full lines are affected.
- I Insert at the beginning of a line. (Same as ^ followed by i.)
- J Join the current line with the next one, supplying appropriate white space: one space between words, two spaces after ". ", and no spaces at all if the first character of the next line is ")". A count causes that many lines to be joined rather than two.
- L Move the cursor to the first non-white character of the last line on the screen. A count moves to that line counting from the bottom. When used with an operator, whole lines are affected.
- M Move the cursor to the middle line on the screen, at the first non-white position on the line.
- N Scan for the next match of the last pattern given to / or ?, but in the reverse direction; this is the reverse of n.
- O Open a new line above the current line and enter input mode.

- P Put the last deleted text back before (above) the cursor. The text goes back as whole lines above the cursor if it was deleted as whole lines. Otherwise the text is inserted just before the cursor. May be preceded by a named buffer specification ("*x*"), to retrieve the contents of the buffer.
- Q Quit from *vi* and enters *ex* command mode.
- R Replace characters on the screen with characters entered, until the input is terminated with ESC.
- S Change whole lines (same as *cc*). A count changes that many lines.
- T Must be followed by a single character; scans backwards in the current line for that character, and if found, places the cursor just after that character. A count is equivalent to repeating the search that many times.
- U Restore the current line to its state before the cursor was last moved to it.
- W Move forward to the beginning of a word in the current line, where a word is a sequence of non-blank characters. A count specifies the number of words to move forward.
- X Delete the character before the cursor. A count repeats the effect, but only characters on the current line are deleted.
- Y Place (yank) a copy of the current line into the unnamed buffer (same as *yy*). A count copies that many lines. May be preceded by a buffer name to put the copied line(s) in that buffer.
- ZZ Exit the editor, writing out the buffer if it was changed since the last write. (Same as the *ex* command *x*.)
- a Enter input mode, appending the entered text after the current cursor position. A count causes the inserted text to be replicated that many times, but only if the inserted text is all on one line.
- b Back up to the previous beginning of a word in the current line. A word is a sequence of alphanumerics or a sequence of special characters. A count repeats the effect.
- c Must be followed by a movement command. Deletes the specified region of text and enters input mode to replace it with the entered text. If more than part of a single line is affected, the deleted text is saved in the numeric buffers. If only part of the current line is affected, the last character to be deleted is marked with a \$. A count is passed through to the move command. If the command is *cc*, the whole of the current line is changed.
- d Must be followed by a movement command. Deletes the specified region of text. If more than part of a line is affected, the text is saved in the numeric buffers. A count is passed through to the move command. If the command is *dd*, the whole of the current line is deleted.

- e Move forward to the next word-end, defined as for *b*. A count repeats the effect.
- f Must be followed by a single character; scans the rest of the current line for that character, and moves the cursor to it if found. A count repeats the find that many times.
- h Move the cursor one character to the left. (Same as *^H*.) A count repeats the effect.
- i Enter input mode, inserting the entered text before the cursor. (See *a*.)
- j Move the cursor one line down in the same column. (Same as *^J* and *^N*.)
- k Move the cursor one line up. (Same as *^P*.)
- l Move the cursor one character to the right. (Same as *<space>*.)
- m Must be followed by a single lower-case letter *x*; marks the current position of the cursor with that letter. The exact position is referred to by *`x*; the line is referred to by *^x*.
- n Repeat the last / or ? scanning commands.
- o Open a line below the current line and enters input mode; otherwise like *O*.
- p Put text after/below the cursor; otherwise like *P*.
- r Must be followed by a single character; the character under the cursor is replaced by the specified one. (The new character may be a newline.) A count replaces each of the following count characters with the single character given.
- s Delete the single character under the cursor and enter input mode; the entered text replaces the deleted character. A count specifies how many characters from the current line are changed. The last character to be changed is marked with a \$, as for *c*.
- t Must be followed by a single character; scans the rest of the line for that character. The cursor is moved to just before the character, if it is found. A count is equivalent to repeating the search that many times.
- u Reverse the last change made to the current buffer. If repeated, will alternate between these two states, thus is its own inverse. When used after an insert which inserted text on more than one line, the lines are saved in the numeric named buffers.
- w Move forward to the beginning of the next word, where "word" is the same as in *b*. A count specifies how many words to go forward.
- x Delete the single character under the cursor. With a count deletes that many characters forward from the cursor position, but only on the current line.

- y Must be followed by a movement command; the specified text is copied (yanked) into the unnamed temporary buffer. If preceded by a named buffer specification, "x, the text is placed in that buffer also. If the command is yy, the whole of the current line is copied (yanked) into the unnamed temporary buffer.
- z Redraw the screen with the current line placed as specified by the following character: <carriage-return> specifies the top of the screen, "." the centre of the screen and - the bottom of the screen. A count may be given after the z and before the following character to specify the new screen size for the redraw. A count before the z gives the number of the line to place in the centre of the screen instead of the default current line.

APPLICATION USAGE

As this editor depends on the optional *Curses* package (see **Volume 3, XSI Curses Interface**), applications should be aware that it may not be present or not function with certain classes of terminal.

SEE ALSO

ex.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

wait – await completion of process

SYNOPSIS

wait [pid]

DESCRIPTION

With no argument, *wait* waits until all background processes (started with &) of the current shell have completed and then reports on abnormal terminations. If a numeric argument *pid* is given and is the process ID of a background process, then *wait* waits until that process has completed. Otherwise, if *pid* is not a background process, *wait* waits until all background processes have completed.

This command operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

APPLICATION USAGE

This is always a shell built-in command.

SEE ALSO

sh, **Volume 2, XSI System Interfaces and Headers**, *wait()*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

NAME

wall – write to all users

SYNOPSIS

wall

DESCRIPTION

The *wall* utility reads a message on the standard input until an end-of-file. It then prints this message on the terminals of all users currently logged-in, preceded by an announcement indicating who sent the message.

The sender must have appropriate privileges to override any protections the users may have invoked, see *mesg*.

APPLICATION USAGE

The *wall* utility is used to send warning messages to all users, typically prior to shutting down the system.

SEE ALSO

mesg, *write*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

wc – word count

SYNOPSIS

OF **wc [-lwc] [file ...]**

DESCRIPTION

The *wc* utility counts lines, words and bytes in the *file* or files named, or in the standard input if no *file* appears. It also keeps a total count for all named files. A word is defined as a maximal string of bytes delimited by a whitespace character.

The options *l*, *w* and *c* may be used in any combination to specify that a subset of lines, words and bytes are to be reported. The default is *-lwc*.

When *files* are specified on the command line, their names will be printed along with the counts.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN **LC_CTYPE** determines which characters are defined as “whitespace” characters.

If **LC_CTYPE** is not set in the environment or is set to the empty string, the value of **LANG** will be used as a default. If **LANG** is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

what – identify SCCS files

SYNOPSIS

what [-s] file...

DESCRIPTION

The *what* utility searches the given files for all occurrences of the pattern that *get* (see *get*) substitutes for %Z% (@(#)) and prints out what follows until the first ", >, newline, \ or null character.

The *what* utility is intended to be used in conjunction with the SCCS command *get*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

There is only one option:

-s Quit after finding the first occurrence of the pattern in each file.

EXIT STATUS

Exit status is:

- 0 any matches were found
- 1 otherwise

EXAMPLES

If the C-language program in file *f.c* contains

```
char ident[] = "@(#)identification information";
```

and *f.c* is compiled to yield *f.o* and *a.out*, then the command

```
what f.c f.o a.out
```

will print

```
f.c:      identification information
          ...
f.o:      identification information
          ...
a.out:    identification information
          ...
```

If only one *file* argument is given, the *filename:* information is not output.

SEE ALSO

get.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

who – who is on the system

SYNOPSIS

```
OF who
OF UN who [ options ] [ file ]
OF who am i
OF who am I
```

DESCRIPTION

The *who* utility can list the user's name, terminal line, login time, elapsed time since activity occurred on the line and the process ID of the command interpreter for each current system user. It examines an implementation-defined database of logged-on users to obtain its information. If *file* is given, that file is examined instead.

The *who* utility with the *am i* or *am I* option identifies the invoking user.

Except for the default *-s* option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With *options*, *who* can list logins, logoffs, reboots and changes to the system clock, as well as other processes spawned by the *init* process. These *options* are:

- u** This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory */dev*. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked *old*. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process ID of the user's login process.
- T** This option is the same as the *-u* option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. The process with appropriate privileges can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed. See *mesg*.
- l** This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.
- H** This option will print column headings above the regular output.
- q** This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.
- p** This option lists any other process which is currently active and has been previously spawned by *init*.

- d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values of the dead process. This can be useful in determining why a process terminated.
- b This option indicates the time and date of the last reboot.
- r This option indicates the current *run-level* of the *init* process.
- t This option indicates the last change to the system clock.
- a This option turns on all options.
- s This option is the default and lists only the *name*, *line* and *time* fields.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**.

Environment Variables

IN *LC_TIME* determines the format and contents of date and time strings output in the "time" column.

If *LC_TIME* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

SEE ALSO

mesg.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

NAME

write – write to another user

SYNOPSIS

write user [terminal]

DESCRIPTION

The *write* utility copies its standard input to the terminal of another user. When first called, *write* sends a message to the user addressed.

The recipient of the message should write back, by typing *write sender-login-id*, on receipt of the initial message. Whatever each user types (except for command escapes, see below) is printed on the other user's terminal, until an end-of-file or an interrupt is sent. At that point *write* writes

EOT

on the other terminal and exits. The recipient can also stop further messages from coming in by executing *mesg n*.

To write to a user who is logged in more than once, the *terminal* argument may be used to indicate which terminal to send to (e.g., *tty00*); otherwise, the first writable instance of the user found in the implementation-defined database of logged-on users is assumed and an informational message is written; see *who*.

A user may deny or grant permission to write by use of the *mesg* utility. Certain commands disallow messages in order to prevent interference with their output. However, if the sender has appropriate privileges, messages can be forced onto a write-inhibited terminal.

If the character *!* is found at the beginning of a line, *write* calls the command interpreter to execute the rest of the line as a command.

ERRORS

The following errors are reported:

- the user addressed is not logged on.
- the user addressed denies write permission, see *mesg*.
- the user's terminal is set to *mesg n* and the recipient cannot respond.
- the recipient changed permission (*mesg n*) after *write* had begun.

SEE ALSO

mesg, *who*.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [options] [command [initial-argument ...]]

DESCRIPTION

The *xargs* utility combines the fixed *initial-argument* or arguments with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the *options* specified.

If *command* is omitted, *echo* is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash quotes the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see *-i*). Options *-i*, *-l* and *-n* determine how arguments are selected for each command invocation. When none of these options are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. When there are conflicts (e.g., *-l* vs. *-n*), the last option has precedence. The recognised *options* are:

*-l*number

Command is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation to the next non-empty line. If *number* is omitted, 1 is assumed. Option *-x* is forced.

-ireplstr

Insert mode: *command* is executed for each line from standard input, taking the entire line as a single argument, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of five arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option *-x* is also forced. {} is assumed for *replstr* if not specified.

-nnumber

Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option *-x* is also invoked, each *number* arguments must fit in the *size* limitation, otherwise *xargs* terminates execution.

-t Trace mode: the *command* and each constructed argument list are echoed to standard error just prior to their execution.

-p Prompt mode: the user is asked whether to execute *command* at each invocation. Trace mode (*-t*) is turned on to print the command instance to be executed, followed by a *?... prompt*. A reply of *y* (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

-x Causes *xargs* to terminate if any argument list would be greater than *size* characters; *-x* is forced by the options *-i* and *-l*. When neither of the options *-i*, *-l* or *-n* are coded, the total length of all arguments must be within the *size* limit.

-ssize

The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If *-s* is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-eofstr

eofstr is taken as the logical end-of-file string. Underscore, *"_"* is assumed for the logical EOF string if *-e* is not invoked. The option *-e* with no *eofstr* coded turns off the logical EOF string capability (underscore is taken literally). The utility *xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

The utility *xargs* will terminate if either it receives a return code of *-1* from, or if it cannot execute, *command*. (Thus *command* should explicitly *exit* with an appropriate value to avoid accidentally returning with *-1*.)

EXAMPLES

1. The following will move all files from directory *\$1* to directory *\$2*, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{} 
```

2. The following will combine the output of the parenthesised commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

3. The user is asked which files in the current directory are to be archived. The files are archived into **arch**; a, one at a time, or b, many at a time.

- a. `ls | xargs -p -l ar r arch`

- b. `ls | xargs -p -l | xargs ar r arch`

4. The following will execute with successive pairs of arguments originally typed as command line arguments:

`echo $* | xargs -n2 diff`

SEE ALSO

echo.

CHANGE HISTORY

First released in Issue 2.

Issue 3

Functionally equivalent to the entry in Issue 2.

NAME

yacc – a compiler-compiler

SYNOPSIS

MV UN yacc [-vdlft] grammar

DESCRIPTION

The *yacc* utility provides a general tool for describing the input to a program. More precisely, *yacc* converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; built-in precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C-language compiler to produce a program **yyparse**. This program must be loaded with the lexical analyser function, *yylex()*, as well as *main()* and *yyerror()*, an error-handling routine. These routines must be supplied by the user (however, see the description of the *yacc* library below); *lex* is useful for creating lexical analysers usable by *yacc*.

If the *-v* option is used, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the *-d* option is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

MV UN If the *-l* option is used, the code produced in **y.tab.c** will not contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

UN Run-time debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, *yacc*'s *-t* option is used, this debugging code will be compiled by default. Independent of whether the *-t* option was used, the run-time debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the run-time debugging code will be smaller and slightly faster.

The yacc library

The *yacc* library **liby.a** facilitates the initial use of *yacc* by providing the routines:

```
main()
yyerror(s)
char *s;
```

These routines may be loaded by using the *-ly* option with *cc*. The *main()* function just calls *yyparse()*. The *yyerror()* function simply prints the string (error message) *s* when a syntax error is detected.

Specifications for yacc

The *yacc* user constructs a specification of the input process; this includes rules that describe the input structure, the code to be invoked when these rules are recognised, and a low-level routine to do the basic input. The *yacc* utility then generates the (integer-valued) function *yyparse()*; it in turn calls *yylex()*, the lexical analyser, to obtain input tokens.

A structure recognised (and returned) by the lexical analyser is called a *terminal symbol*, here referred to as a *token* (literal characters must also be passed through the lexical analyser, and are also considered tokens). A structure recognised by the parser is called a *non-terminal symbol*. *Name* refers to either tokens or non-terminal symbols.

Every specification file consists of three sections: *declarations*, *grammar rules* and *programs*, separated by `%%`. The declarations and programs sections may be empty. If the latter is empty, then the preceding `%%` mark separating it from the rules section may be omitted.

Spaces, tabs and newlines are ignored, except that they may not appear in names or multi-character reserved symbols. Comments are enclosed in `/* ... */` and may appear wherever a name is valid.

Names may be of arbitrary length, made up of letters, `"."`, `"_"` and non-initial digits. Upper- and lower-case letters are distinct. Names beginning in *yy* should be avoided, since the *yacc* parser uses such names.

A literal consists of a character enclosed in single quotes. The C-language escape sequences (e.g., `'\n'`) are recognised.

Declarations

The following declarators may be used in the declarations section:

%token

Names representing tokens must be declared; this is done by writing

```
%token name1 name2 ...
```

in the declarations section. Every name not defined in this section is assumed to represent a non-terminal symbol. Every non-terminal symbol must appear on the left-hand side of at least one grammar rule.

%start

The *start symbol* represents the largest, most general structure described by the grammar rules. By default, it is the left-hand side of the first grammar rule; this default may be overridden by declaring:

```
%start symbol
```

%left

%right

%nonassoc

Precedence and associativity rules attached to tokens are declared using these keywords. This is done by a series of lines, each beginning with one of the keywords *%left*, *%right* or *%nonassoc*, followed by a list of tokens.

All tokens on the same line have the same precedence level and associativity; the lines are in order of increasing precedence or binding strength. *%left* denotes that the operators on that line are left associative and *%right* similarly denotes right-associative operators. *%nonassoc* denotes operators that may not associate with themselves. (A token declared using one of these keywords need not be declared by *%token* as well.)

%prec

Unary operators must, in general, be given a precedence. In cases where a unary and binary operator have the same symbolic representation, but need to be given different precedences, the keyword *%prec* is used to change the precedence level associated with a particular grammar rule. *%prec* appears immediately after the body of the grammar rule, before the action or closing semicolon (see **Grammar Rules** below), and is followed by a token name or a literal. It causes the precedence of the grammar rule to become that of the following token name or literal.

%union

By default, the values returned by actions and the lexical analyser are integers. Other types, including structures, are supported: the *yacc* value stack is declared to be a union of the various types of values desired. The utility *yacc* keeps track of types, and inserts appropriate union-member names so that the resulting parser will be strictly type checked. The declaration is done by including a statement of the form:

```
%union {  
    body of union  
}
```

Alternatively, the union may be declared in a header and a **typedef** used to define the variable *YYSTYPE* to represent this union. The header must be included in the declarations section by using a **#include** construct within *%{* and *%}* (see below). Union members must be associated with the various names. The construction *<name>* is used to indicate a union-member name; if this follows one of the keywords *%token*, *%left*, *%right* and *%nonassoc*, the union-member name is associated with the tokens listed.

%type

This key word is used to associate union member names with non-terminals, in the form:

```
%type <ntype> a b ...
```

Other declarations and definitions can appear in the declarations section, enclosed by the marks *%{* and *%}*. These have global scope within the file, so that they may be used in the rules and programs sections.

Grammar Rules

The rules section is comprised of one or more grammar rules. A grammar rule has the form:

`A : BODY ;`

A represents a non-terminal name, and *BODY* represents a sequence of zero or more names and literals. The colon and the semicolon are *yacc* punctuation. If there are several successive grammar rules with the same left-hand side, the vertical bar, "|", can be used to avoid rewriting the left-hand side; in this case the semicolon must occur only after the last rule. The *BODY* part may be empty to indicate that the non-terminal symbol matches the empty string.

The null character (0 or '\0') should not be used in grammar rules.

With each grammar rule, the user may associate actions to be performed each time the rule is recognised in the input process. These actions may return values and may obtain the values returned by previous actions. In addition, the lexical analyser can return values for tokens, if desired.

An action is an arbitrary C-language statement, and as such can do input or output, call subprograms and alter external variables. An action is one or more statements enclosed by { and }. Certain pseudo-variables can be used in the action: a value can be returned by assigning it to \$\$; the variables \$1, \$2, ... refer to the values returned by the components of the right-hand side of a rule, reading from left to right. By default, the value of a rule is the value of the first element in it. Actions may occur in the middle of a rule as well as at the end; an action may access the values returned by symbols (and actions) to its left, and in turn the value it returns may be accessed by actions to its right.

Internal rules to resolve ambiguities are:

1. In a shift/reduce conflict, the default is to do the shift.
2. In a reduce/reduce conflict, the default is to reduce by the grammar rule that occurs earlier in the input sequence.

In addition, the declared precedences and associativities (see **Declarations Section** above) are used to resolve parsing conflicts as follows:

1. A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the %prec keyword is used, it overrides this default. Some grammar rules may have no precedence and associativity.
2. When there is a reduce/reduce conflict, or there is a shift/reduce conflict and either the input symbol or the grammar rule has no precedence and associativity, then the two rules given above are used.
3. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have precedence and associativity associated with them, then the conflict is resolved in favour of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the

associativity is used; left associative implies reduce, right-associative implies shift, and non-associative implies error.

Conflicts resolved by precedence are not counted in the shift/reduce and reduce/reduce conflicts reported by *yacc*.

The token name *error* is reserved for error handling. This name can be used in grammar rules; in effect, it suggests places where errors are expected and recovery might take place. When an error is encountered, the parser behaves as if the token *error* was the current look-ahead token and performs the action encountered. The look-ahead token is then reset to the token that caused the error. If no special error rules have been specified, the processing halts when an error is detected.

In order to prevent a series of error messages, the parser, after detecting an error, remains in error state until three tokens have been successfully read and shifted. If an error is detected when the parser is already in error state, no message is given, and the input token is quietly deleted.

The statement

```
yyerrok;
```

in an action resets the parser back to its normal mode; it may be used if it is desired to force the parser to believe that an error has been fully recovered from.

The statement

```
yyclearin;
```

in an action is used to clear the previous look-ahead token; it may be used if a user-supplied routine is to be used to find the correct place to resume input.

Programs

The programs section may include the definition of the lexical analyser *yylex()* and any other functions; for example those used in the actions specified in the grammar rules.

The function *yylex()* is an integer-valued function which returns the *token number*, representing the kind of token read. If there is a value associated with that token, it should be assigned to the external variable *yytval*. The parser and *yylex()* must agree on these token numbers in order for communication between them to take place. The numbers may be chosen by *yacc* or chosen by the user. In either case, the **#define** construct of C-language is used to allow *yylex()* to return these numbers symbolically. If the token numbers are chosen by *yacc*, literals are given the numerical value of the character in the local character set and other names are assigned token numbers starting at 257.

A token may be assigned a number by following its first appearance in the declarations section with a non-negative integer. Names and literals not defined in this way retain their default definition. All token numbers must be distinct.

The end of the input is marked by a special token called the *endmarker*. The endmarker must have token number 0 or negative. (These values are not valid

for any other token.) All lexical analysers should return 0 or negative as a token number upon reaching the end of their input. If the token up to but excluding the endmarker forms a structure which matches the start symbol, the parser accepts the input. If the endmarker is seen in any other context, it is an error.

This utility operates in an 8-bit transparent manner, see **Section 2.2.1, Eight Bit Transparent Utilities**. The *yacc* utility provides 8-bit transparency for characters contained in character strings, character constants and comment strings. The support of 8-bit characters in *names* is implementation defined.

Environment Variables

IN *LC_CTYPE* determines which characters are defined as letters and digits (for *name* fields).

If *LC_CTYPE* is not set in the environment or is set to the empty string, the value of *LANG* will be used as a default. If *LANG* is not set or is set to the empty string, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contain an invalid setting, the command will behave as if none of the variables had been defined.

ERRORS

The number of reduce/reduce and shift/reduce conflicts is reported on the standard error output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

SEE ALSO

lex.

CHANGE HISTORY

First released in Issue 2.

Issue 3

The operation of this utility in an 8-bit transparent manner has been noted.

The operation of this utility in an internationalised environment has been described.

Withdrawn Interfaces

Some of the interfaces in this issue have been marked as withdrawn. Various factors may have contributed to the decision to withdraw an interface. The reasons for withdrawal of an interface are documented on the relevant pages.

If a migration path exists, advice is given to application developers regarding alternative means of obtaining similar functionality. This information may be found in the **APPLICATION USAGE** sections on the relevant pages.

Withdrawn interfaces may still exist on conformant implementations. However, they will be dropped from the next issue of the Guide. Application writers should not use interfaces marked as withdrawn.

NAME

as – common assembler (**WITHDRAWN**)

SYNOPSIS

as [-o objfile] [-m] [-V] file

APPLICATION USAGE

The *cc* utility is the recommended interface to the assembler. The *as* utility itself may not be present on all X/Open systems.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

cpp – the C-language preprocessor (**WITHDRAWN**)

SYNOPSIS

cpp [options] [ifile] [ofile]

APPLICATION USAGE

The recommended way to invoke the C-language preprocessor utility is through the *cc* utility.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

file – determine file type (**WITHDRAWN**)

SYNOPSIS

file file...

file -f lfile

APPLICATION USAGE

This utility has been withdrawn because it uses heuristics to determine the filetype and its accuracy should therefore not be relied upon.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

ld – link editor for object files **(WITHDRAWN)**

SYNOPSIS

ld [options] file...

APPLICATION USAGE

The recommended way to invoke the link editor is through the *cc* utility.

SEE ALSO

cc.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

lorder – find ordering relation for an object library (**WITHDRAWN**)

SYNOPSIS

lorder file...

APPLICATION USAGE

The link editor phase of *cc* is capable of multiple passes over an archive in the *ar* format and does not require that *lorder* be used when building an archive.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

mknod – build FIFO file (**WITHDRAWN**)

SYNOPSIS

mknod name p

APPLICATION USAGE

This utility has been withdrawn as it has been superseded by *mkfifo*.

SEE ALSO

mkfifo.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

news – print news items (**WITHDRAWN**)

SYNOPSIS

news [-a] [-n] [-s] [item ...]

APPLICATION USAGE

This utility has been withdrawn as it was of an administrative nature and so it tended to vary between implementations and was therefore of no use to applications.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

passwd

Withdrawn Utilities

NAME

passwd – change login password (**WITHDRAWN**)

SYNOPSIS

passwd [name]

APPLICATION USAGE

This utility has been withdrawn as it cannot be implemented on secure implementations, and was, in any case, not usable by applications.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

prof – display profile data (WITHDRAWN)

SYNOPSIS

prof [-tcan] [-ox] [-g] [-z] [-m mdata] [prog]

APPLICATION USAGE

This utility has been withdrawn as the operation cannot be supported consistently across all implemenations. Implementations may provide a local means of profiling.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been withdrawn.

NAME

shl – shell layer manager (**WITHDRAWN**)

SYNOPSIS

shl

APPLICATION USAGE

This utility has been withdrawn as it will be replaced by functionality based on job control which is being introduced with an optional status in **Volume 2, XSI System Interfaces and Headers**.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

NAME

size – print section sizes of object files **(WITHDRAWN)**

SYNOPSIS

size [-o] [-x] [-V] file...

APPLICATION USAGE

This utility has been withdrawn as the information it provided was entirely system dependent and frequently misleading. It was therefore of no use to portable applications.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been withdrawn.

NAME

su – become a user with appropriate privileges or another user (**WITHDRAWN**)

SYNOPSIS

su [-] [name [arg ...]]

APPLICATION USAGE

This utility has been withdrawn as it may not exist on systems providing a secure environment.

CHANGE HISTORY

First released in Issue 2.

Issue 3

This page has been marked as withdrawn.

Index

/dev/null: 31
/dev/tty: 31
@modifier: 27
[n, m): 22
[n, m]: 22
.: 11, 14, 16, 18, 115, 186, 232, 304, 311
..: 11, 14, 16, 18
_POSIX_JOB_CONTROL: 15
_POSIX_NO_TRUNC: 17
abandoned: 184, 242
abnormal termination: 245
absolute pathname: 9
access control:
 file: 12
 file, additional: 12
 file, alternate: 12
access mode: 9
access permissions: 14, 16, 192
 file: 12
access time: 127
account:
 login: 28
address space: 9, 19
admin: 37-40, 92-93, 136, 212
ANSI X3.159: 9
append: 42, 103, 112, 177
application programs: 1
appropriate privileges: 9
ar: 24, 42-43, 76, 114, 187, 309, 323
archive: 42-43, 75, 98, 185, 187, 195, 261-262, 323
argument list: 114, 116-117, 191, 236, 307-308
ARG_MAX: 30
array: 26, 49-50, 221-223
as: 1-2, 4-5, 7-8, 23-24, 33, 44, 58, 155, 319
ASCII: 26, 53, 69, 90-92, 119, 131
assembler: 58
asynchronous terminal: 32
at: 7, 32, 44-45
atime: 127, 129
awk: 24, 46-48, 50-52, 142
background: 180, 233, 300
background process group: 9
banner: 53, 154
basename: 54
batch: 44-45
baud: 83, 252
binary: 92, 98, 130, 136, 266, 312
binding: 312
bits: 14, 64, 66, 127, 158, 252-253, 275, 289
bitwise: 162
block: 32, 90-91, 127, 257, 262
block special file: 9
bound: 26
break: 49, 82, 149, 238, 253, 310
bss: 195
buffer: 90, 101-107, 112-113, 115-118, 121, 245, 260, 293-295, 297-299, 307
byte: 68, 74, 91, 94, 99, 139, 158-159, 197-198, 222-223, 240, 257, 260
C standard: 9
C-language: 16, 49, 58-60, 62, 86, 144, 221, 223, 226, 303, 320
C-language program: 144, 149, 186
C-program: 58, 60, 130, 144, 149, 187
cal: 55
calendar: 55-56
caller: 62
cancel: 24, 153-154, 282, 291
cat: 24, 57, 80, 82-84, 199, 204
catch: 143
category: 27
cc: 24, 58-60, 63, 86, 98, 144, 149-150, 170, 172, 183, 186, 188, 251, 297, 310, 319-320
cd: 24, 33, 61, 75, 82, 169, 233, 238, 241-242, 262
cflow: 62-63
character: 9
 null: 16
character array: 144
character set:
 portable filename: 14, 28, 30

- character sets: 9
- character special file: 10
- character string: 26, 134
- chdir: 61, 67, 169
- chgrp: 24, 66
- child process: 10, 19
- children: 67
- chmod: 24, 64-66, 73, 127, 275
- chown: 24, 66
- chroot: 67
- clock tick: 10
- close: 4, 252
- cmp: 24, 68, 71, 96-97
- codeset: 15
- col: 69
- collation: 26-27
- comm: 68, 71, 96, 142
- command: 1, 10, 15, 32-33, 61, 100, 192, 216, 233, 265-266
- command interpreter: 1, 10, 15, 46, 77, 80, 82, 85, 103, 105-106, 109, 117-119, 124, 128, 138, 164, 169, 174, 176, 180, 183, 192, 201, 204, 226, 231, 266, 270, 285-286, 304, 306
- command prompt: 15
- common: 4, 7, 25, 39, 71, 97, 141, 169, 183, 187, 192, 226, 242
- compatibility: 69, 149-150
- compiler: 58-59, 186, 310
- connection: 10, 81-82, 252
- constants: 8, 49, 222-223, 225
- controlling process: 10
- controlling terminal: 9-10
- coordinated universal time: 11
- core file: 221
- cp: 24, 72
- cpio: 24, 73-75, 128, 263
- cpp: 58, 151, 163, 320
- crontab: 45, 77-78
- csplit: 79-80
- ctime: 87-88, 127
- cu: 81-83
- current directory: 44, 54, 56, 74, 99, 101, 120, 135, 157, 192, 196, 201, 237-238, 242, 277, 279, 284, 286, 309
- current position: 80, 115, 292-294, 298
- current process: 162
- current:state: 189
- current time: 15, 282
- current user: 74, 282
- current value: 117, 240, 275
- current working directory: 17, 22, 43, 224, 239
- cut: 62, 85, 287
- cxref: 86
- database: 157, 202, 291
 - user: 26, 28
- date: 24, 40, 44, 56, 87-88, 131, 134-135, 175, 182-183, 185, 187, 206, 209, 220, 268, 305, 308
- dd: 87, 90-91, 134, 297
- debugging: 178, 310
- decimal: 62, 66, 68, 86, 101-102, 127, 162, 195, 197, 222-223, 227, 233, 246, 269
- default: 37-38, 40, 43, 46, 48, 58, 62, 64, 68, 74, 77, 79, 81, 83, 86-88, 90, 101-103, 109, 114, 116-120, 131, 134, 137, 141, 144, 146-147, 149, 153, 155, 157, 161-162, 164-165, 167, 173-175, 177-180, 185-186, 192, 194, 201-204, 206-207, 209-210, 213, 221-224, 227, 229, 234-236, 238, 245-246, 249-250, 258, 262, 265, 268, 276, 279-280, 284, 291, 299, 302, 304-305, 308, 310-314
- default action: 146
- default timezone: 29
- defined: 26
- delay: 254
- deliver: 260
- delta: 37-41, 92-93, 131-136, 209, 212, 219-220, 277, 289-290
- development: 2
- DEVELOPMENT: 35, 37, 62, 86, 92, 98, 131, 144, 149, 160, 182, 195, 209, 219-221, 251, 277, 289
- device: 9-10, 69, 81, 94, 158, 213, 252, 262, 265
 - terminal: 22
- device ID: 10
- df: 94
- diagnostics: 40
- diff: 24, 68, 71, 93, 95, 97, 286, 309
- digits: 44, 87-88, 160, 162, 231, 246, 269, 275, 311

Index

- dircmp: 97
- directory: 11-13, 15, 17-18, 22, 25-26, 28, 37, 43-44, 54, 56, 60-61, 64, 67, 72, 74-75, 77, 82, 92, 94-95, 97, 99, 101, 120-121, 127-128, 131, 133, 135-136, 157-159, 167, 169, 171, 177-179, 185, 190, 192, 196, 198, 209, 216-217, 219-221, 224, 237-239, 241-242, 261-262, 265, 277, 279-280, 284, 286, 291, 304, 308
 - current working: 22
 - empty: 11
 - entry: 11
 - parent: 16
 - root: 20
 - working: 22
- directory entry: 11
- directory stream: 11
- dirname: 54, 284
- dis: 98
- document: 70, 236
- documentation: 4, 226
- domain: 280
- dot: 11, 14, 16, 18, 113, 115, 178-179, 186, 232, 304, 311
- dot-Dot: 11
- dot-dot: 14, 16, 18
- du: 99
- EBCDIC: 90-91
- echo: 24, 45, 53, 81, 83-84, 100, 170, 187, 233, 237-238, 254-255, 291, 307-309
- ed: 24, 95-96, 101-102, 104-107, 121, 170, 178, 227, 229-230, 254
- effective group ID: 11, 15
- effective user ID: 11, 22
- egrep: 24, 109-110, 138
- element: 50, 77, 141, 313
- empty directory: 11
- empty string: 11, 23, 26, 28, 48-49, 52, 161, 201, 265, 313
- encoded: 87, 130
- end-of-file: 37, 68, 147, 165, 178-179, 236, 239, 289, 301, 306, 308
- env: 111
- environ: 26
- environment: 26-27, 43-44, 60-61, 77, 111, 117, 120, 153, 157, 165, 167-168, 173, 175, 177-178, 184, 192, 196, 202-204, 221, 234, 236-239, 259, 291
 - environment variable: 26
 - environment variables: 27, 44, 77, 177, 192
 - EOF: 164, 226, 255, 285
 - EOL: 255
 - epoch: 11
 - ERASE: 254-255
 - ex: 112, 120, 170, 172, 174, 291-295, 297, 299
 - exec: 1, 33, 128-129, 192, 231, 237-238, 240, 242
 - execute mode: 182
 - execute permission: 159, 237
 - executes: 22
 - execution environment: 111, 175, 177
 - execution time: 213, 310
 - exit: 7, 50, 60, 93, 105, 107, 117, 125, 128, 148, 150, 162, 164-165, 170, 175, 181, 231-233, 238-240, 242, 265, 268, 272, 274, 285, 289, 304-305, 308
 - expr: 24, 50, 124-126, 265-266
 - expression: 50
 - extended security controls: 12
 - extension: 198
 - external variable: 314
 - false: 24, 265, 272, 293
 - feature: 5, 102, 105, 107, 109, 131
 - feature test macro: 12
 - fgrep: 24, 109-110, 138
 - FIFO special file: 12, 14, 18, 127, 191
 - fildest: 265
 - file: 8-9, 11-17, 20-23, 31, 37-40, 42-46, 48, 50, 52, 54, 56-58, 60, 64-68, 71-75, 77-82, 85-86, 90-91, 94-95, 97-99, 101, 103-107, 109, 127-128, 130, 137, 139, 141, 144-147, 149-151, 153, 157-160, 162-165, 167-180, 182-183, 185-187, 192-193, 195-199, 201-204, 206-207, 213, 217, 219-220, 227, 229, 245, 247-251, 257, 261-262, 264-265, 268, 273, 275-280, 284-285, 289-290, 301-304, 306, 308, 310-312, 315, 319, 321-324
 - block special: 9, 12
 - character special: 10, 12
 - FIFO special: 12
 - regular: 12
 - file access mechanism:

- additional: 12
- alternate: 12
- file access permissions: 11-12, 14
- file description: 14, 16, 43
- file descriptor: 13, 16, 235-236, 241, 265
- file group class: 13-14
- file hierarchy: 13
- file mode: 13-14
- file name: 135, 234-235, 239
- file offset: 14, 16
- file other class: 14
- file owner class: 13-14
- file permission: 14, 127
- file permission bits: 14
- file serial number: 14
- file space: 99, 153
- file system: 14, 20
- file times update: 14
- file types: 157-158
- filename: 14, 17-18, 28, 37-38, 45-46, 62, 71, 90, 103, 105-106, 109, 115, 127, 131, 133-134, 137, 151, 153, 157-158, 160, 167, 170-171, 173-180, 182, 185, 193, 201-202, 204, 207, 211, 222, 224, 226, 245, 279, 286-287, 292, 303
- find: 24, 61, 73, 75-76, 127-129, 149, 153, 167, 249, 295, 298
- flag: 38-40, 92-93, 116, 127, 131-132, 134-135, 179, 198, 227, 237, 241, 246, 251, 259, 278, 291
- floating point number: 16
- flush: 255
- fold: 171
- foreground process group: 15
- foreground process group ID: 15
- fork: 19
- FORTTRAN: 258
- fstat: 15
- gencat: 24, 130
- generation: 232, 239, 252
- get: 38-39, 41, 63, 79, 92-93, 99, 131-133, 135-136, 152, 157-158, 170, 175, 212, 219-220, 277, 286-287, 290, 303
- gets: 287
- GMT: 87
- gname: 127
- grep: 24, 85, 110, 137-138, 201, 230
- group database: 192
- group file: 66
- group ID: 15
 - effective: 11, 15
 - real: 15, 20
 - saved set: 15
 - supplementary: 15, 21
- group name: 66
- handle: 27, 271
- hang up: 252
- hardware: 9, 21, 252, 259, 276
- header: 74, 164, 167-168, 170-173, 175, 177-178, 180, 193, 204, 207, 312
- hexadecimal: 98, 195, 197, 222-223
- hierarchy: 13, 75, 127-128
- highlight: 4
- HOME: 26, 61, 77, 169, 238, 240
- home directory: 291
- I/O: 14, 25, 31, 198, 205, 252
- iconv: 24, 139
- id: 38, 140, 153-155, 306
- ID:
 - process: 19
 - process group: 19
- IEEE: 4
- IEEE P1003.2: 1, 4, 6, 76, 263
- IEEE std. 1003.1: 18, 28
- IFS: 234-235, 239-240
- implementation dependent: 158-159
- implementation specific: 165
- IN: 43, 50, 71-73, 75-76, 87-88, 107, 110, 125, 128, 137, 141, 156, 159, 165, 180-181, 207, 214, 217, 230, 245, 247, 261-262, 270, 278, 280-282, 302, 305, 315
- inclusion: 62, 186
- init: 304-305
- initial working directory: 26, 67
- input character: 69, 293
- input parity checking: 253
- input queue: 253
- int: 50, 63
- integer: 13, 50, 62, 114, 120, 124, 127, 202, 222, 225, 308, 311, 314
- interface: 1-2, 9, 24, 58, 144, 187, 192, 205, 256, 280, 283, 285, 287, 319
- interpreter: 1, 15, 46, 77, 80, 85, 103, 105-106, 109, 117-119, 124, 128, 138, 174,

Index

- 176, 180, 183, 192, 201, 204, 226, 231, 266, 286, 306
- interval: 244
- INTR: 254-255
- IS 8859/1: 21, 76, 165, 181, 263, 281, 285, 287
- ISO: 4, 139
- ISO 8859/1: 18
- job control: 15, 282
- join: 104, 115, 141
- julian: 30
- kill: 24, 82, 143, 215, 241
- LA: 4-5, 55, 87, 104, 249
- LANG: 26-28
- language operation: 26
- LC_ALL: 26
- LC_COLLATE: 27
- LC_CTYPE: 26-27
- LC_MONETARY: 26-27
- LC_NUMERIC: 26-27
- LC_TIME: 26-27
- ld: 42-43, 58, 60, 195, 251, 322
- lex: 24, 62-63, 144-147, 149, 186-188, 310, 315
- lifetime: 20
- line: 37, 39-40, 44, 46-53, 56, 62-63, 68-70, 74, 77, 79-83, 85, 90, 92, 94, 98, 100, 109, 111-122, 128, 130-132, 134-135, 137, 139, 141, 144, 148-150, 153-155, 157, 164-165, 167-168, 171, 173, 175-179, 183-184, 193-194, 201-204, 206-207, 209-210, 212, 217, 219-222, 224-229, 239, 245-247, 249-250, 252, 254-256, 260, 270, 277-278, 280, 283-285, 287, 289-290, 304-309, 312
- line numbers: 116, 121
- link: 15, 42, 58-60, 72-73, 75, 217, 242, 262, 322
- link count: 15
- link-editor: 58
- lint: 62-63, 149-151
- ln: 24, 72, 149-151, 194
- locale: 23, 26-27
- lock: 136
- login account: 28
- login name: 66, 92, 127, 135, 152, 177, 213, 277, 279, 284, 304
- LOGNAME: 28, 77
- logname: 152, 220, 308
- lorder: 43, 323
- lp: 24, 132, 153-155
- lpstat: 24, 153-155
- ls: 24, 75-76, 157, 179, 181, 308-309
- LV: 109
- m4: 160-162
- macro: 118, 121-122, 160-161, 182-185, 187, 265
 - feature test: 12
- magic: 119, 121
- mail: 24, 45, 139, 153-154, 164-165, 167-168, 172, 175-176, 180-181, 234, 280, 284, 287
- mailx: 24, 167-170, 172-175, 177-179, 181
- main: 63, 80, 147, 211, 221, 245, 310
- make: 32, 39, 42, 85, 90, 151, 182-187, 199, 207, 220, 248, 252, 279-280, 293
- marked for update: 15
- mask: 240, 275
- maximum portability: 4
- mesg: 189, 301, 304, 306
- message catalogue: 15
- message catalogue descriptor: 16
- MIN: 255
- mkdir: 24, 190
- mkfifo: 191, 324
- mknod: 324
- mode: 16
- modem connection: 252
- modem lines: 252
- modification time: 75, 234, 261-262, 282
- monetary: 26-27
- mtime: 127
- multi-byte: 139
- MV: 5
 - mv: 24, 54, 72-73
- MV: 98, 118, 144, 175, 195, 251, 258
- mv: 308
- MV: 310
- name=value: 26
- NAME_MAX: 14, 17
- NaN: 16
- native language: 15
- network: 167, 174, 177, 179, 276
- newgrp: 33, 192

- news: 325
- NGROUPS_MAX: 21
- nl: 193-194
- NL: 197
- nl: 255
- nm: 195
- no-op: 71
- nodename: 276
- nohup: 196
- NUL: 197, 254
- null:
 - character: 16
 - pointer: 16
 - string: 16
- null byte: 223
- null character: 16, 201
- null pathname: 18
- null pointer: 16
- null string: 16, 93, 124
- object file: 42, 98, 186-187, 195, 251
- octal: 64, 68-69, 98, 100, 127, 195, 197, 222-223, 231, 240, 269, 275
- octal value: 240
- od: 197
- odd parity: 81
- OF: 5, 44-45, 53, 55, 68, 71, 87, 94, 97, 99, 104, 140, 153, 155, 157-158, 189, 197-198, 213, 228, 249, 255, 257, 276, 280, 302, 304
- OP: 5, 58, 112, 221, 291
- open file: 14, 16, 265, 289
- open file description: 14, 16
- OPEN_MAX: 13
- operating system: 4, 276
- optimiser: 58
- OPTIONAL: 2, 98, 167, 192, 221
- orphaned process group: 16
- output characters: 269
- output modes: 253
- output queue: 204
- owner: 13-14, 40, 64, 66, 72, 74, 135, 157-158, 198-199, 219, 261, 275
- pack: 24, 198
- parameter: 28, 117, 231-236
- parent: 22
- parent directory: 16, 190
- parent process: 16, 19, 21
- parent process ID: 17
- parity: 81, 252-253, 255
- passwd: 127, 326
- password: 66, 192, 326
- paste: 85, 201
- PATH: 28, 61, 77, 233-235, 237-238, 240-241
- path prefix: 17
- pathname: 17-18, 22
 - component: 17
 - relative: 20
 - resolution: 17
- pathname component: 14, 17
- PATH_MAX: 17-18
- pause: 202
- pcat: 24, 198
- permission: 14, 39, 61, 64-65, 128, 135, 158-159, 189-190, 217, 237, 275, 306
- permission bits: 14
 - file: 12-13
- permissions:
 - file access: 12
- pg: 24, 173, 175, 178-179, 181, 202-205
- PI: 5, 46, 57-59, 95, 98-99, 149, 153-155, 164-165, 167, 195, 197, 202, 206, 213, 245, 257, 262
- pid: 143, 300, 304
- pipe: 14, 18, 45, 50, 100, 127, 173, 178-179, 201, 203, 205, 231, 260, 265
- pipeline: 205, 231, 262, 287
- pointer:
 - null: 16
- popen: 1, 33
- port: 149
- portable: 5, 18, 42, 75, 149-150, 199, 257
- portable filename: 18
- portable filename character set: 14, 18, 28, 30
- portable pathname: 18
- pr: 24, 32, 201, 206-207
- prefix: 17, 28, 54, 79, 135, 147, 186
- preprocessor: 58
- prime meridian: 29
- primitives: 265
- printf: 50, 87
- privilege: 19
- process: 19
 - arguments: 30

Index

- child: 19
- lifetime: 19
- parent: 16, 19
- system: 21
- process group: 9, 19, 21, 31, 143
- process group ID: 19
- process group ID: 19
- process group leader: 19
- process group lifetime: 19
- process ID: 19, 300
 - parent: 17
- process lifetime: 19
- prof: 60, 327
- program message: 15
- prompt: 39, 81, 92-93, 101, 104, 112, 121, 164, 168, 179, 202, 204, 234-235, 308
- prototype: 8
- prs: 41, 93, 136, 209-212, 219, 290
- ps: 24, 143, 213-214
- PS1: 234-235, 240
- PS2: 234-235, 240
- puts: 184
- pwd: 24, 61, 216, 233, 239, 242
- pwd.h: 26, 28
- queue: 45, 204, 253, 280
- radix: 27, 162
- range: 9, 23, 77, 104, 114-119, 132, 168, 221, 227-228
- read by group: 64
- read by others: 64
- read by owner: 64
- read-only file system: 20
- real group ID: 15, 20
- real time: 267
- real user ID: 22
- receipt: 167, 178, 204, 287, 306
- receiver: 252
- red: 24, 101
- region: 222, 292, 295, 297
- regular expression:
 - extended: 47, 109
 - extended internationalised: 47, 109
 - simple: 101, 124, 137, 203, 227
 - simple internationalised: 101, 124, 137, 203, 227
- regular file: 12, 14, 20, 265
- relative pathname: 20
- release: 37-39, 133-134, 175, 276
- rename: 72-74
- return value: 239
- rm: 24, 73, 129, 187, 217
- rmdel: 93, 209, 219
- rmdir: 24, 217
- root directory: 17-18, 20, 67
- running process: 67
- sact: 220, 277
- save: 117, 172, 178-180, 227
- saved set-group-ID: 15, 20
- saved set-user-ID: 20, 22
- SCCS: 21
- scheduler: 77, 155
- sdb: 58, 60, 221-223, 225-226
- security: 81, 192, 280, 287
- sed: 24, 108, 110, 138, 227
- seeking: 127, 236
- select: 5, 27, 74, 227, 252, 254
- session: 21, 175
- session leader: 21
- session lifetime: 21
- set user: 64
- setlocale: 26-27
- sh: 1, 24, 33, 45, 54, 61, 78, 80, 85, 100, 108-109, 111, 117, 125-126, 174, 180, 188, 192, 231-232, 242, 266, 272, 286-287
- shell: 23, 33, 45, 74, 169, 173-174, 176, 178-180, 187, 192, 231-242, 279, 300, 328
- shl: 328
- SIGHUP: 143, 196, 242, 252
- SIGINT: 143, 167, 178, 182, 204, 237, 241-242, 253, 292, 295
- SIGKILL: 143, 242
- signal: 21, 104, 106, 112, 143, 191, 196, 204, 224-225, 240, 242, 252-253
- SIGQUIT: 143, 182, 196, 204, 237, 241-242
- SIGTERM: 143, 241-242
- simple-command: 231, 237
- size: 87, 90, 93, 127, 158, 174, 195, 199, 202, 204, 206, 223, 240, 245, 252, 262, 265, 291-292, 299, 308, 310, 329
- size field: 158
- slash: 14, 17-18, 21-22, 157-158, 178, 232
- sleep: 24, 244
- sort: 24, 71, 141-142, 158, 245-248, 278
- special characters: 124, 171, 270, 286, 293,

- 297
- special file:
 - block: 9
 - block : 12, 127, 158, 265
 - character: 10, 14, 22
 - character : 9, 12, 22, 74, 127, 158, 260, 265
 - FIFO: 12, 18
- special files: 25, 31, 74-75, 191, 260
- spell: 45, 249
- split: 2, 49, 79-80, 105, 235, 250
- sprintf: 50
- square brackets: 145, 214, 266
- standard C: 60, 149
- standard error: 44-45, 77, 98, 132, 195-196, 251, 267, 308, 315
- standard input: 37, 44-46, 57, 68-69, 71-72, 74, 77, 79, 82, 85, 90, 92-93, 95, 106, 109, 118-119, 131, 133, 137, 139, 141, 144, 153, 160, 164-165, 182-183, 193, 197, 201-202, 206, 209, 217, 219-220, 224-227, 231, 235-236, 239, 241, 245, 249-250, 252, 257, 260, 262, 264, 269, 273-274, 277-278, 284, 287, 289-290, 293, 301-302, 306-308
- standard output: 4, 45-46, 49, 51, 53-57, 59, 62, 69, 74, 77, 82, 86, 90, 92-93, 100, 109, 124, 132-133, 137, 139-140, 144, 148, 152-153, 160, 176, 193, 196, 198-199, 201, 204, 206-207, 209, 211-212, 216, 227-229, 231-232, 235-236, 245, 249, 259-260, 262, 269, 273, 276-278, 280, 284-287, 289, 293
- START: 253
- stat: 15
- static data: 62
- stdio: 110
- stdio.h: 110
- stream: 139, 146, 157, 160-162
- string: 26-27, 38, 46-54, 87, 93, 98, 101-102, 105, 107, 109, 117-122, 124, 131, 134, 141, 144-146, 160-162, 168, 170, 174, 176-177, 179-180, 183, 185, 193, 201-202, 209-210, 222, 224-226, 228-229, 233-234, 240-241, 246, 254, 261, 265, 269-270, 278, 280, 286-287, 289, 295, 302, 308, 310, 313
- empty: 11
- null: 16
- strip: 43, 251, 253
- structure: 13, 20, 25, 40, 101, 221, 223, 311, 315
- structure members: 221
- stty: 24, 82-84, 252, 259
- su: 330
- subroutines: 145
- subscript: 221
- suffix: 28, 44, 54, 58, 185-187, 199, 203
- sum: 40, 51, 64, 206, 257, 267
- supplementary group ID: 15, 21
- suspend: 15, 244
- sync: 90, 160
- sys/stat.h: 13-14
- system: 21
- system default: 155, 167, 245
- system process: 21
- tabs: 69, 77, 83, 95, 115, 119-122, 160, 184, 205-207, 228, 236, 254-255, 258-259, 278, 302, 307, 311
- tail: 24, 260, 262
- tape: 75, 91, 262
- tar: 24, 76, 261-262
- tee: 24, 264
- TERM: 29
- terminal: 22
- terminal device: 22, 265
- terminal input: 204
- terminal interface: 205
- termination: 50, 116, 164, 174, 177, 232, 235, 240, 245, 305
- territory: 15
- test: 24, 232, 240, 265-266, 274
- time: 40, 44-45, 69, 75, 87-88, 101, 107, 127, 131, 134-136, 143-144, 151, 157-158, 165, 168, 177, 183, 202, 204, 206, 209, 213-214, 220-221, 232, 234-235, 237, 244, 249
- TIME: 255
- time: 260, 262, 267-268, 282, 304-305, 309-310, 313
- time-related fields: 15
- timezone: 29, 87-88
 - default: 29
- title: 154

Index

- TMPDIR: 29
- touch: 174, 183, 268
- tr: 24, 42, 61-62, 66-67, 69, 74, 77, 81, 86, 92, 97, 100, 124, 141, 152, 193, 196, 198, 201-202, 206, 209, 216-217, 219-221, 226-227, 232-233, 235, 238, 244-245, 249-251, 264, 267, 269-272, 275, 277-279, 284, 286-287, 300-302, 306, 326, 330
- trailer: 207
- translate: 269
- translation: 122
- transmission: 81, 122, 279, 284
- troff: 249
- true: 24, 128, 192, 214, 244, 265, 272
- tsort: 273
- tty: 24, 31, 274
- type arguments: 80
- TZ: 29, 87-88
- ulimit: 44, 240
- umask: 24, 44, 240, 275
- UN: 5, 43, 59, 81, 87, 94, 99, 144, 153-155, 158, 164-165, 167, 182, 185, 193, 195, 202, 209, 213, 221, 251-252, 257-258, 268-269, 279-280, 282, 284, 286-287, 304, 310
- uname: 24, 83-84, 127, 276
- undefined: 62, 195, 255
- underscore: 62, 160, 231, 233, 308, 311
- underscores: 231
- unset: 220, 277
- uniq: 24, 71, 142, 278
- unique: 13-14, 97, 153, 157, 248, 286
- units: 94, 99, 158-159, 197, 223, 240, 257, 260
- unpack: 24, 198
- unsigned: 197, 203-204, 222
- unspecified: 5, 38
- upper-case: 253
- user database: 26, 28
- user ID: 22, 66
 - effective: 11, 22
 - real: 22
 - saved set-: 22
- utilities: 2-4, 10, 32
- utility: 4-5, 7, 37, 42, 44-47, 53-60, 62, 66-69, 71, 73, 77, 79-81, 85-86, 90-92, 94-95, 97-99, 101, 111-112, 120, 124, 127, 130-131, 137, 140-141, 144, 148-149, 151-155, 160-161, 164, 167, 183, 189-193, 196, 204, 206, 209, 213, 217, 220-221, 227, 230-231, 242, 244-245, 249, 251, 256-259, 261-262, 269, 271-272, 274, 276-280, 282-287, 301-304, 306, 319-321
- uucp: 24, 82, 84, 178, 279-280, 282, 284-286
- uulog: 24, 279-280
- uname: 24, 279-280, 284
- uupick: 24, 284-285
- uustat: 24, 280-282, 285, 287-288
- uuto: 24, 284-285
- uux: 24, 280-281, 285-287
- val: 160, 289-290
- variable:
 - environment: 26
- variables: 26
- version: 38, 101, 118, 131, 175, 179, 195, 199, 219, 231, 251, 276
- vi: 112, 118, 123, 178, 180-181, 291, 297
- view: 186, 199
- wait: 24, 231, 240-241, 261, 300
- wall: 301
- wc: 24, 302
- what: 41, 95, 97, 131, 134, 136, 184, 192, 199, 207, 210, 212, 214, 303
- who: 4, 24, 39, 64, 135, 220, 301, 304, 306
- window: 122, 202-204, 291-293
- WITHDRAWN: 3, 319-330
- working directory: 17, 22, 26, 61, 67, 216, 224, 239
- write: 45, 64-65, 101, 106, 115-117, 123, 128, 135, 158, 175, 189-190, 217, 241, 262, 275, 279, 297, 301, 304, 306
- write by group: 64
- write by others: 64
- write by owner: 64
- xargs: 307-309
- yacc: 24, 62-63, 144, 147, 149, 186-188, 310-314
- zombie process: 22

